**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ**
**КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ**

«На правах рукопису»
УДК 004.4

«До захисту допущено»
Завідувач кафедри
_____
_____
(підпис) (ініціали, прізвище)
“___”_____20__
р.

# Магістерська дисертація

зі спеціальності (спеціалізації) 121, інженерія програмного забезпечення (інженерія програмного забезпечення комп’ютерних систем) _____
(код і назва спеціальності)

на тему: Інтегрована система відображення стану мережевої інфраструктури в режимі реального часу

Виконав: студент 2 курсу, групи ІТ-83мп
(шифр групи)

Решетицький Олександр Вікторович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент кафедри АУТС, к.т.н, доцент Полторак В.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент в.о. зав. каф. СПіСКС, д.т.н, доцент Романкевич В.О. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

# РЕФЕРАТ

Магістерська дисертація присвячена розробці інтегрованої системи відображення стану мережевої інфраструктури в режимі реального часу.

Магістерська дисертація містить 116 аркушів пояснювальної записки, 14 рисунків, 60 таблиць, 8 креслеників та 19 бібліографічних посилань на використані літературні джерела.

Актуальність даної роботи полягає у тому, що внаслідок зростання вимог до мережевої інфраструктури виникає потреба в засобах спостереження за її станом. Існуючі засоби представляють собою інтегровані рішення в складі систем моніторингу мереж, проте представляють обмежені можливості для візуалізації.

Об’єктом дослідження виступає складна мережева інфраструктури, її складові елементи та зв’язки між ними. Предметом дослідження є спостереження за змінами стану мережевої інфраструктури та їх відображення в зручному для користувача вигляді.

Елементом науково-практичної новизни у даній роботі виступає легкість інтеграції системи. Архітектурою системи передбачено механізм сервісів-стрімерів, які будуть надсилати інформацію про зміни в стані мережевої інфраструктури в чергу повідомлень. На основі отриманих повідомлень буде побудовано граф стану мережевої інфраструктури. Інформацію про зміни стану мережі буде відображено користувачу в режимі реального часу.

Компоненти розробленої системи були впроваджені ПрАТ «Будінвестмережа-2» у якості частини системи спостереження за станом локальної мережі.

Ключові слова: мережева інфраструктура, сервіси-стрімери, черга повідомлень, граф стану, відображення змін в реальному часі.

# SUMMARY

Master’s thesis is dedicated to development of integrated real-time system for displaying state of network infrastructure.

Master’s thesis contains 116 sheets of explanatory note, 14 pictures, 60 tables, 8 drawings and 19 bibliographic references on used literary sources.

The relevance of this work lies in the fact that growing requirements for network infrastructure require tools for observing its state. Existing tools are represented by integrated solutions within network monitoring systems, but they present limited visualization capabilities.

The object of research is the complex network infrastructure, its components and connections between them. The subject of research is monitoring of network infrastructure state changes and its display in a user-friendly manner.

An element of scientific and practical novelty in this work is systems ease of integration. System architecture provides a mechanism of streaming services that will send information on state changes for network infrastructure to message queue. State graph for network infrastructure will be constructed based on the received messages. Changes to network state will be displayed to the user in real-time mode.

Components of the developed system were implemented by PJSC «Budinvestmerezha-2» as part of local network monitoring system.

Keywords: network infrastructure, streaming services, message queue, state graph, real-time change display.

**Національний технічний університет України**
**«Київський політехнічний інститут**
**імені Ігоря Сікорського»**

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Автоматики та управління в технічних системах
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 121 інженерія програмного забезпечення комп'ютерних систем
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І.Ролік
(підпис) (ініціали, прізвище)
«___»_____20__ р.

**ЗАВДАННЯ**
**на магістерську дисертацію студенту**
Решетицькому Олександру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Інтегрована система відображення стану мережевої інфраструктури в режимі реального часу

науковий керівник дисертації Полторак Вадим Петрович, к.т.н., доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації 17 грудня 2019 р.

3. Об'єкт дослідження складні мережеві інфраструктури

4. Предмет дослідження (вихідні дані для магістерської дисертації за освітньо-професійною програмою) спостереження за змінами стану мережевої інфраструктури та їх відображення

5. Перелік завдань, які потрібно розробити сформувати набір вимог до системи, розглянути існуючі рішення, реалізувати механізм отримання даних з висхідних систем, розробити програмну систему для відображення змін стану мережевої інфраструктури

6. Орієнтовний перелік ілюстративного (графічного) матеріалу

діаграма прецедентів, діаграма класів, діаграма синхронізації, діаграма компонентів, ER-діаграми сервісів системи

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації*

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |

9. Дата видачі завдання 24 жовтня 2018 р.

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Строк виконання етапів магістерської дисертації | Примітка |
|---|---|---|---|
| 1 | Порівняльний аналіз існуючих рішень | 02.09.19 – 06.09.19 | |
| 2 | Визначення основних вимог до системи | 09.09.19 – 13.09.19 | |
| 3 | Розроблення сценаріїв використання системи | 16.09.19 – 20.09.19 | |
| 4 | Вибір технологій для розробки | 23.09.19 – 27.09.19 | |
| 5 | Розроблення структурної схеми | 30.09.19 – 04.10.19 | |
| 6 | Розроблення ER-діаграми | 07.10.19 – 11.10.19 | |
| 7 | Реалізація бізнес-логіки | 14.10.19 – 18.10.19 | |
| 8 | Розроблення інтерфейсу користувача | 21.10.19 – 25.10.19 | |
| 9 | Розроблення стартап-проекту | 28.10.19 – 02.11.19 | |
| 10 | Оформлення текстового матеріалу | 04.11.19 – 25.11.19 | |
| 11 | Оформлення графічного матеріалу | 26.11.19 – 02.12.19 | |
| 12 | Подача дисертації на перевірку | 03.12.19 – 18.12.19 | |

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База Даних

SSE – Server Sent Events

WAL – Write-Ahead Log

Oplog – Operations Log

DSL – Domain Specific Language

AQL – Arango Query Language

GQL – Graph Query Language

VNF – Virtual Network Function

EAV – Entity-Attribute-Value

ПЗ – Програмне Забезпечення

# ВСТУП

Внаслідок розвитку комп’ютерних мереж поступово виникла проблема підтримки її стану. Програмне забезпечення для моніторингу мереж стежить за пристроями, трафіком та серверами в корпоративних чи освітніх мережах та повідомляє мережевих адміністраторів, коли все піде не так. Це ключова зброя в наборі інструментів мережевого менеджера для вирішення проблем із мережею.

Мережева інфраструктура представляє собою сукупність різного устаткування та програмного забезпечення, які формують середовище для передачі даних та забезпечує виконання інформаційними системами, що користуються мережею, основних функцій, таких як збирання, надсилання, перетворення, накопичення, зберігання та оброблення інформації[1]. Мережева інфраструктура також надає засоби автоматизації основних інформаційних процесів, що пов’язані з отриманням інформації про стан мережі. Інформаційний процес представляє собою процес збору, підготовки, передачі, обробки, перетворення та використання даних[2].

Моніторинг мережі охоплює все, від простих домашніх сценаріїв перевірки роботи хостів, до розповсюджених корпоративних програм із агентами на різних хостах та клієнтах.

Програмне забезпечення для моніторингу мережі може стежити за мережевим трафіком та використанням пропускної здатності. Він може перевірити, чи є важливі мережеві компоненти, такі як комутатори, маршрутизатори та сервери, в придатному до роботі стані чи ні.

Однією з задач моніторингу є візуалізація стану мережевої інфраструктури. Сюди входить відображення елементів мережі, зв’язки між ними, відображення різних параметрів пристроїв, метрик, що відображають характеристики стану елементу тощо. Для вирішення даної задачі існує безліч різноманітних інструментів з різним функціоналом.

Досить часто таке рішення представляється у вигляді компонента комплексної системи моніторингу. Проте дані інструменти досить часто мають певний ряд

недоліків, які запобігають максимізації їх ефективного використання. Одним з таких недоліків є обмеженість функціоналу, що може проявлятися у малих опціях контролю рівня відображення елементів, обмеженні можливості конфігурації тощо.

З іншого боку, існують окремі системи, що повністю спрямовані на аспекті відображення стану мереж. Дані інструменти надають більше можливостей власне візуалізації порівняно з інтегрованими рішеннями, однак мають свій набір проблем, з якими може стикатися користувач, такі як наприклад прив’язка до певних протоколів для збору даних про мережу, або обмеження лише статичною візуалізацією без оновлення в разі змін стану мережі.

Метою магістерської дисертації є розроблення програмної системи відображення стану мережевої інфраструктури. Розроблювана система повинна представляти широкі можливості з візуалізації графу мережі, перегляду стану елементів мережевої інфраструктури, зв’язки між ними. Граф мережі повинен будуватися автоматично та оновлюватися при зміні стану елементів мережевої інфраструктури. Система повинна надавати механізми інтеграції з існуючими системами моніторингу для відокремлення способу отримання даних про стан мережевої інфраструктури від функціоналу побудови графу мережі та його відображення.

# 1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Системи, що дозволяють відображати стан мережевої інфраструктури досить часто представляються у вигляді додатку до основної системи моніторингу. Саме тому порівняння з даними типами систем буде проведено лише в контексті можливостей відображення графу мережевої інфрастуртури.

Для порівняння було обрано наступні існуючі системи побудови графу мережі: Zabbix, TheDude та LanState.

## 1.1 Система моніторингу Zabbix

Zabbix – вільна система моніторингу мережевих служб та станів мережевої системи. Архітектура Zabbix включає в себе три базові частини[3]:

- сервер для координації виконання перевірок, формування перевірочних запитів та накопичення статистики;
- агенти для проведення перевірок на стороні зовнішніх хостів;
- веб-клієнта для організації управління усією системою.

Zabbix представляє собою потужну та широку за функціональністю систему моніторингу, яка представляє користувачу ще півдесятка функцій, які дозволять ще більше спростити спостереження за мережею, такі як моніторинг стану веб-сайтів за допомогою автоматичного виконання сценарію на кшталт «зайти в систему, переглянути нові повідомлення та вийти».

Для відображення логічної структури мережі можна створити карту мережі, що відображає вузли мережі та зв’язки між ними. Створення карти мережі в Zabbix відбувається лише в ручному режимі, що є мабуть найбільшим недоліком даної системи.

При створенні карти мережі система надає можливість користувачу визначити ряд параметрів карти, такі як власник, ім’я карти, відображення стану елементів

мережі, посилання на елементи мережі тощо. Екран початкового налаштування карти мережі представлено на рисунку 1.1.

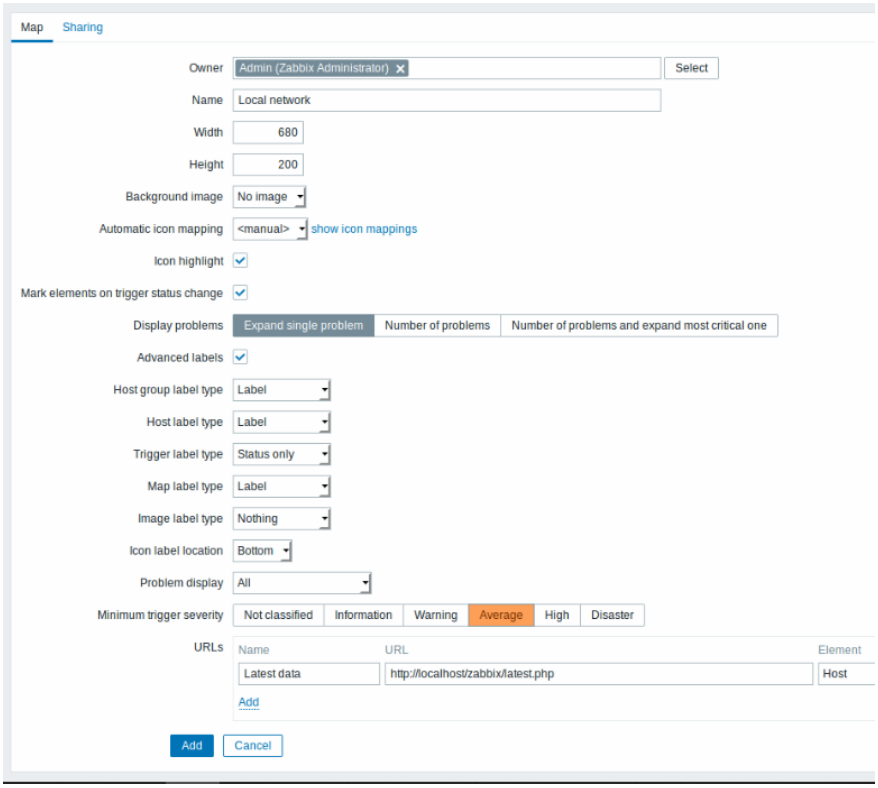


Рисунок 1.1 – Меню створення мережі в Zabbix

Після створення мережі, користувач повинен в ручному режимі перетягнути необхідні елементи з переліку доступних. Кожен з елементів можна налаштувати, вказавши тип елемента, його ім’я, іконку тощо. При обранні елемента, відкриється діалог вибору елемента, в якому можна власне здійснити його налаштування.

Розставивши всі елементи, наступним кроком є побудова зв'язків між елементами. Для додавання зв'язку необхідно обрати обидва елементи та натиснути на кнопку «Додати» у підменю «Зв'язок». Для зв'язків також надаються широкі опції з налаштування, такі як підпис, тип зв’язку, колір, індикатори зв’язку тощо. Приклад меню налаштування зв’язку наведено на рисунку 1.2.

Система представляє зручний інтерфейс з гнучкими можливостями налаштування, проте неможливість побудови мережевої карти в автоматичному режимі є значним недоліком даної системи.

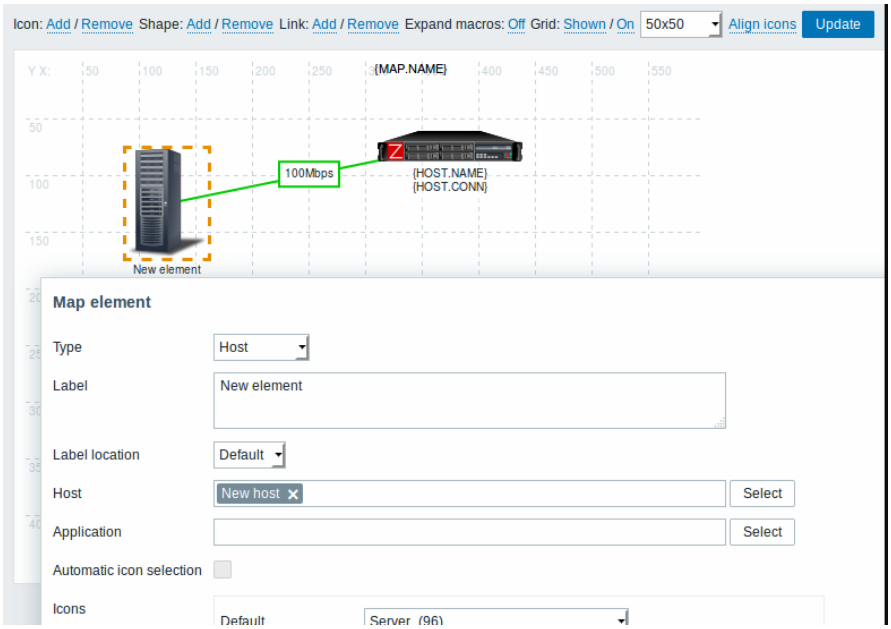


Рисунок 1.2 – Меню налаштування зв’язку в Zabbix

## 1.2 Програма для адміністрування та моніторингу мережі LANState

LANState – програма компанії 10-Strike, розроблена для адміністрації та моніторингу мереж. Розробниками системи заявлено наступний набір переваг порівняно з аналогічними рішеннями на ринку[4]:

- сканування мережі, розпізнавання різного типу пристроїв;
- побудова графічної карти мережі з наочною деталізацією;
- робота одночасно з декількома картами;
- розпізнавання нових пристроїв в мережі;
- web-інтерфейс в якості клієнтської частини;
- для роботи програми не потребується встановлення жодних компонентів на робочі станції та сервери.

Система представляє зручний та ергономічний інтерфейс для огляду карти мережі. Приклад інтерфейсу системи представлено на рисунку 1.3.

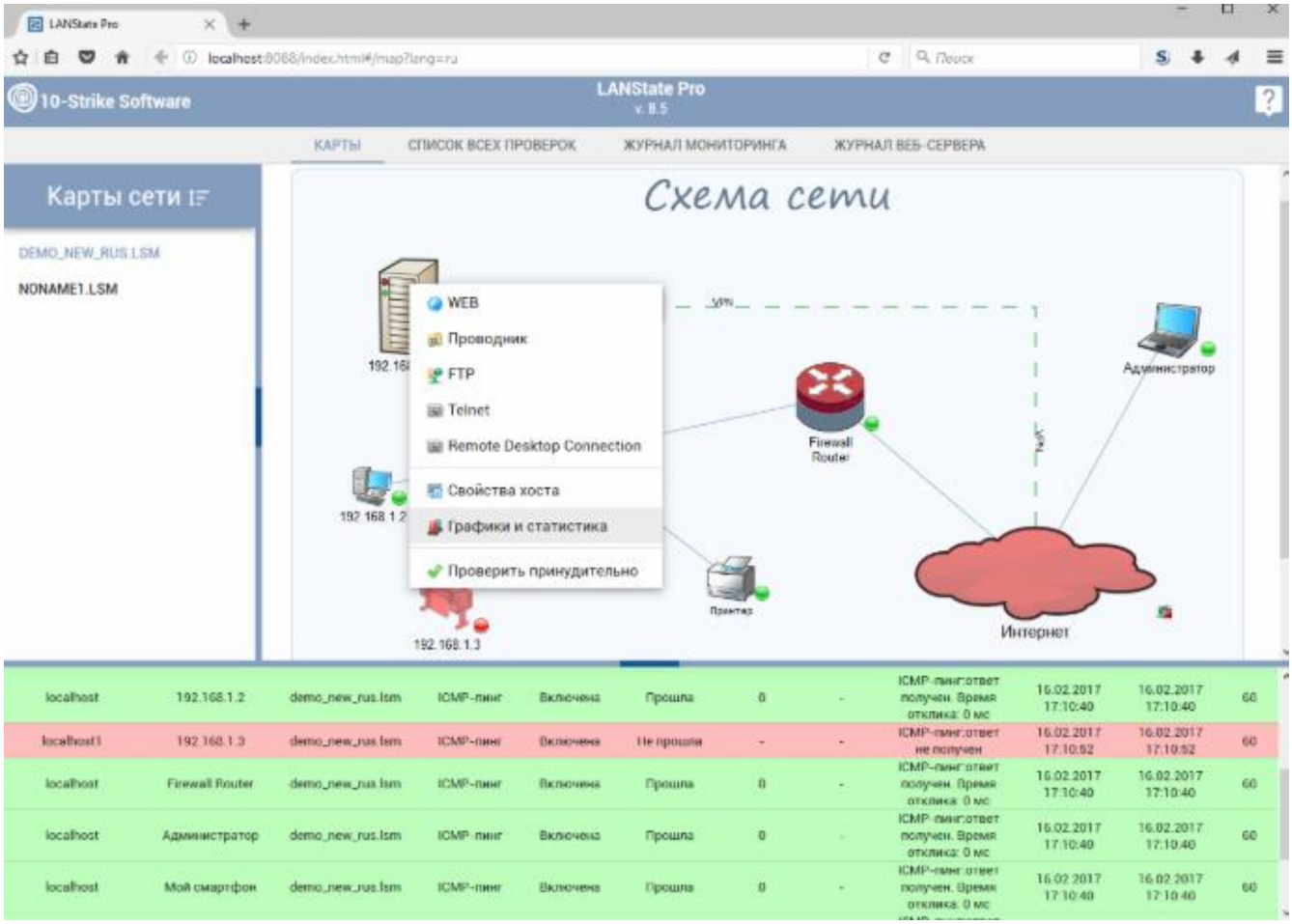


Рисунок 1.3 – Інтерфейс програми LanState

Одним з недоліків системи є обмежені можливості зі збору даних та їх візуалізації. Наприклад, для відображення метрик елементу мережі в наявності лише перевірка стану елемента (up/down).

## 1.3 Система моніторингу The Dude

Система моніторингу The Dude є розробкою компанії Mikrotic. Основним призначенням даної системи є моніторинг пристроїв компанії Mikrotic, проте система також може працювати з іншими пристроями, хоча дана інформація не заявляється офіційно на сайті програми (також відсутній перелік підтримуваних пристроїв)[5].

Функціонал, що надає система, є досить широким і включає в себе автоматичне сканування мережі та відображення на карті мережі, моніторинг пристроїв в мережі та зв’язки між ними, сповіщень про їх стан, відображення пристроїв в графічному вигляді, можливість графічно відображати зв’язки між пристроями та багато іншого.

Клієнтська частина реалізована у вигляді desktop-застосунку, набір підтримуваних платформ обмежується Windows. Інтерфейс системи представлено на рисунку 1.4.

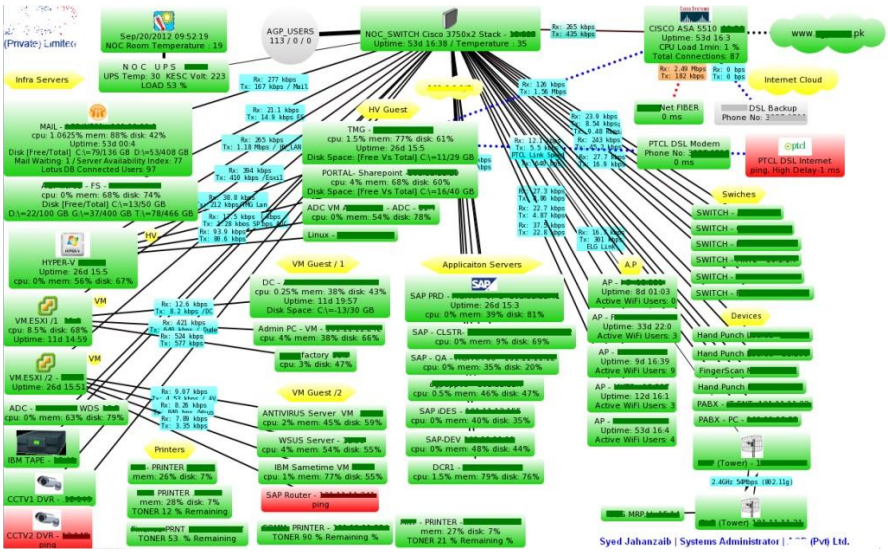


Рисунок 1.4 – Інтерфейс системи моніторингу TheDude

Серед знайдених недоліків системи можна виділити не зовсім дружелюбний до користувача графічний інтерфейс та обмежена підтримка платформ для клієнтської частини.

## 1.4 Порівнювальна характеристика аналогів

Для порівняння аналогів та розроблюваної системи (nvt) було виділено ряд критеріїв, яким повинна відповідати система. Як було зазначено на початку розділу, у порівнянні будуть використовуватися лише можливості систем в контексті

системи, що відображує актуальний стан мережевої інфраструктури. Порівняльна характеристика аналогів представлена в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

| | nvt | Zabbix | The Dude | LanState |
|---|---|---|---|---|
| Відображення оновлень графу в реальному часі | + | - | + | + |
| Автоматична побудова мережевої карти | + | - | + | + |
| Зручний користувацький інтерфейс | + | + | - | + |
| Кроссплатформеність | + | +/- (відсутня версія для ОС Windows) | +/- (відсутня версія для Unix-подібних ОС) | +/- (відсутня версія для Unix-подібних ОС) |
| Широкі можливості відображення даних | + | + | + | - |

## 1.5 Висновки до розділу

В процесі дослідження існуючих аналогів були розглянуті існуючі системи візуалізації мереж у вигляді мережевих карт, досліджено їх особливості, переваги та недоліки.

Система моніторингу Zabbix попри існуючі переваги, такі як широкі можливості конфігурації, має великий недолік у вигляді неможливості автоматизувати процес побудови карти мережі.

LanState володіє цілим рядом переваг, що роблять її найкращим рішенням з представлених аналогів. Автоматична побудова карти мережі, оновлення карти в реальному часі, ергономічний користувацький інтерфейс та інші переваги дозволяють рекомендувати систему, особливо якщо основним пріоритетом стоїть зручна візуалізація стану мережі. Проте ця система все ж має деякі недоліки, що включають відсутність клієнтів для Unix-подібних ОС та обмежені можливості зі збору та відображення даних про елементи карти.

Система TheDude, як і Zabbix, володіє широким переліком можливостей для здійснення моніторингу та відображення карти мережі. Однак, користувацький інтерфейс клієнта залишає бажати кращого, виділяючись нагромадженням елементів та неінтуїтивною організацією елементів меню.

При розробці системи nvt була визначена доцільність її розробки на основі вже існуючих на ринку рішень. Враховуючи результати аналізу аналогів, було виділено набір бажаних критеріїв яким повинна задовольняти система, а саме, зручний інтерфейс, автоматична побудова карти мережі, наявність клієнтів для всіх основних платформ, відображення оновлень стану в мережі в режимі реального часу, надання широких можливостей зі збору даних про елементи в мережі.

# 2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

Вимоги до системи дозволяють визначити властивості, якості та функції системи. За умови наявності правильно та чітко сформульованих вимог полегшується виділення логічних етапів реалізації продукту, встановити пріоритет при реалізації функцій програмного забезпечення, виділити найбільш важливі функції на початкових етапах реалізації системи. Вимоги до системи поділяються на два види – функціональні та нефункціональні. Функціональні вимоги дозволяють встановити які функції система повинна реалізовувати, а функціональні ставлять вимоги до того, як саме ці функції повинні виконуватись.

## 2.1 Формування функціональних вимог

Для розроблюваної системи було визначено наступний перелік функціональних вимог:

- представити засоби інтеграції з іншими системами;
- забезпечити можливість налаштування елементів графу;
- надати можливість регулювання рівня відображення елементів;
- надати можливість ієрархічного конфігурування елементів мережі;
- надати можливість визначення підтипів елементу мережі для виділення спільної конфігурації;
- надати можливість групування елементів за групами;
- надати можливість визначення атрибутів для елементів графу;
- надати можливість визначення обчислюваних атрибутів для елементів графу;
- надати можливість налаштування посилань між елементами на основі обчислювальних атрибутів;
- надати можливість створення посилань на основі шляху в графі між 2-ма елементами;
- надати можливість налаштування запитів для отримання графу;

- надати механізм для синхронізації даних з висхідною системою;
- забезпечити механізм оновлення конфігурації для сервісів системи;
- забезпечити зручний для використання інтерфейс;
- надати можливість вибору вхідної точки для відображення графу мережі;
- представити механізм фільтрації при виборі вхідної точки для відображення графу мережі;
- дозволити при перегляді графу масштабувати вигляд;
- забезпечити можливість згортання-розгортання груп при перегляді графу;
- забезпечити перенесення посилань для елементів-замінників;
- надати можливість перегляду значень для визначених в конфігурації атрибутів елементу;
- забезпечити відображення кількості сповіщень про проблему для елемента мережі на основі даних з систем моніторингу;
- надати можливість перегляду сповіщень про проблему для елемента мережі;
- забезпечити оновлення відображуваного графу у випадку зміни стану в висхідних системах.

## 2.2 Формування нефункціональних вимог

Для даної системи було визначено наступний набір нефункціональних вимог:

- для побудови сервісів-стрімерів повинно бути представлене рішення, що уніфікує підхід до їх розробки;
- при виконанні процедури синхронізації даних, система повинна видалити всі застарілі вузли та посилання;
- при встановленні каналу сповіщень між клієнтом та сервером клієнт повинен надсилати повідомлення про активність за інтервалом 60 секунд;
- клієнтська частина система повинна бути доступною для всіх основних операційних систем;
- система повинна перевіряти запити на обхід графу, що налаштовуються для побудови посилань;

- час на обробку однієї події про оновлення системи не повинен переважати 60 секунд;

- час на побудову клієнтською частиною графу не повинен переважати 240 секунд;

- система не повинна бути прив'язаною до конкретного постачальника хмарних послуг;

- якщо конфігурація не містить налаштувань для певного елементу мережі, повідомлення такого типу елементів будуть ігноруватися;

- час на отримання інформації про кількість сповіщень про проблеми для елементу графу не повинен переважати 30 секунд.

## 2.3 Висновки до розділу

В даному розділі, були визначені необхідні вимоги до системи, які в свою чергу були поділені на функціональні та нефункціональні.

Функціональні вимоги включають в себе вимоги, що зосередженні на вирішені проблем систем-аналогів. Наприклад, автоматичне оновлення відображення графу дозволить користувачу в реальному часі спостерігати за змінами стану мережевої інфраструктури. Система повинна надавати широкий набір можливостей з конфігурації процесу побудови графу стану висхідних систем.

Для нефункціональних вимог було обрано ті вимоги, що пов’язані з якісним виконанням вимог системи, а саме визначений інтервал для перевірки активності з’єднання між клієнтом та сервером, час на обробку події про оновлення стану мережевої інфраструктури, час на побудову клієнтом графу та інші часові обмеження для основного фунціоналу системи.

# 3 СЦЕНАРІЇ ВИКОРИСТАННЯ СИСТЕМИ

Сценарії використання системи є важливою складовою проектування системи. Вони визначають основні сценарії поведінки програмного забезпечення при взаємодії з користувачем і зосередження на виконання певного завдання. Визначення прецедентів системи встановити чіткі алгоритми поведінки системи за різних діях користувача. Отже, вже на стадії проектування будуть детально сформовані основні функції системи.

На основі визначених функціональних вимог були визначені та описані основні сценарії використання системи. Їх зв’язки та зв’язки з акторами зображено на діаграмі прецедентів у додатку А.

## 3.1 Опис прецедентів системи

Відповідно до визначених у попередньому розділі вимог, були встановлені основні актори системи:

- користувач;
- адміністратор системи.

Головним актором при взаємодії з системою є користувач. На початку роботи з системою користувачу необхідно обрати вхідну точку, з якої буде побудовано граф. Для спрощення даного процесу передбачено панель фільтрації для зменшення вибірки за необхідним критерієм.

Адміністратор системи відповідає за підтримку конфігурації елементів мережевої інфраструктури. На стартовому екрані він обирає необхідний тип елементу через ієрархічне меню елементів. Потім він здійснює необхідну конфігурацію для певного елементу мережі.

Детальний опис прецедентів системи зображено на таблицях 3.1-3.11.

Таблиця 3.1 – Опис прецеденту №1

| | |
|---|---|
| Назва | Перегляд конфігурації елементу графа |
| ID | 1 |
| Опис | Адміністратор бажає виконати налаштування елементів топології |
| Актори | Адміністратор системи |
| Організаційні переваги | Налаштовано необхідну конфігурацію елементів топології |
| Частота використання | 100% |
| Причина виникнення | Адміністратору необхідно виконати налаштування конфігурації елементів топології |
| Передумова | Адміністратор зайшов у клієнт для налаштування конфігурації |
| Постумова | Нова конфігурація елементів відповідає правилам валідації |
| Головний шлях виконання | 1. Зі стартового екрану необхідно обрати необхідний тип елементу графу;<br>2. Виконати налаштування основних параметрів типу обравши вкладку «General» на сторінці налаштування типу;<br>3. Виконати налаштування атрибутів типу обравши вкладку «Attributes» на сторінці налаштування типу;<br>4. Натиснути кнопку «Save» для збереження конфігурації. |

Продовження таблиці 3.1

| | |
|---|---|
| Альтернативний шлях виконання | 1. Для пошуку необхідного типу адміністратор може скористатися панеллю пошуку яка буде надавати можливі типи опираючись на значення пошукового рядку;<br>2. Якщо адміністратор бажає додати новий тип, йому необхідно скористатися пунктом «New» в панелі опцій ієрархічного меню;<br>3. Для видалення типу, адміністратору необхідно обрати необхідний тип та скористатися пунктом «Delete» в панелі опцій ієрархічного меню. |
| Виключення | - |

Таблиця 3.2. – Опис прецеденту №2

| | |
|---|---|
| Назва | Налаштування користувацьких запитів для отримання графу |
| ID | 2 |
| Опис | Адміністратор бажає надати користувацький запит для побудови графа залежно від типу стартового елемента |
| Актори | Адміністратор системи |
| Організаційні переваги | При побудові графу мережевої інфраструктури буде використано запит з конфігурації |
| Частота використання | 15% |

Продовження таблиці 3.2

| | |
|---|---|
| Причина виникнення | Адміністратор не влаштовує стандартний алгоритм формування графу |
| Передумова | Адміністратор обрав необхідний тип елементу графа для налаштування та перейшов на сторінку його налаштування |
| Постумова | Описаний адміністратором є правильно сформованим запитом мови AQL |
| Головний шлях виконання | 1. При потраплянні на сторінку налаштування необхідного типу необхідно перейти на вкладку «General»;<br>2. В полі «Query» вказати запит для отримання графу;<br>3. Натиснути кнопку «Save» для збереження конфігурації. |
| Альтернативний шлях виконання | - |
| Виключення | - |

Таблиця 3.3 – Опис прецеденту №3

| | |
|---|---|
| Назва | Налаштування атрибутів елементу графа |
| ID | 3 |
| Опис | Адміністратор бажає налаштувати атрибути елементу графа |
| Актори | Адміністратор системи |
| Організаційні переваги | Значення налаштованих в конфігурації графа елементів використовуються при відображенні графу в клієнті nvt-fe |
| Частота використання | 30% |

Продовження таблиці 3.3

| | |
|---|---|
| Причина виникнення | Адміністратор бажає налаштувати атрибути елементу графу |
| Передумова | Адміністратор обрав необхідний тип елементу графа для налаштування та перейшов на сторінку його налаштування |
| Постумова | Налаштовані атрибути мають відповідати правилам валідації атрибутів |
| Головний шлях виконання | 1. При потраплянні на сторінку налаштування необхідного типу необхідно перейти на вкладку «Attributes»;<br>2. В контекстному меню обрати пункт «New»;<br>3. При переході на сторінку заповнити загальні властивості про атрибут на вкладці «Main» |
| Альтернативний шлях виконання | 1. Якщо атрибут було визначено як обчислюваний, необхідно також налаштувати параметри адаптеру на вкладці «Calculate Info». |
| Виключення | - |

Таблиця 3.4 – Опис прецеденту №4

| | |
|---|---|
| Назва | Налаштування посилання на зовнішній елемент |
| ID | 4 |

Продовження таблиці 3.4

| | |
|---|---|
| Опис | Адміністратор бажає налаштувати зв’язок між елементами різних висхідних систем |
| Актори | Адміністратор системи |
| Організаційні переваги | Між елементами різних систем буде на результуючому графі буде відображено зв’язок згідно налаштувань обчислюваного атрибута |
| Частота використання | 20% |
| Причина виникнення | Адміністратору необхідно налаштувати відображення зв’язку між елементами двох різних висхідних систем |
| Передумова | Адміністратор обрав необхідні типи елементу графа для налаштування зовнішнього зв’язку |
| Постумова | Конфігурація адаптеру обчислюваних атрибути повинна вказувати на існуючі атрибути типів – учасників зв’язку |
| Головний шлях виконання | 1. Користувач повинен перейти на сторінку типу, з якого буде спрямоване висхідне посилання, згідно кроків, описаних у прецеденті №1;<br>2. При потраплянні на сторінку налаштування типу перейти на вкладку «Attributes»;<br>3. Виконати загальне налаштування атрибуту згідно прецеденту 3.4; |

Продовження таблиці 3.4

| | |
|---|---|
| Головний шлях виконання | 4. На вкладці «Calculate Info» для поля «Adapter» вибрати з випадаючого списку значення «External Object Adapter», в полі «external» вказати значення true, в полі «externalAttribute» вказати значення атрибуту, через який можна отримати значення елементу, до якого буде проведено посилання;<br>5. Виконати кроки 2-3 на типі елемента, до якого буде проведено зовнішнє посилання;<br>6. На вкладці «Calculate Info» для поля «Adapter» вибрати з випадаючого списку значення «External Object Adapter», в полі «external» вказати значення false, в полі «externalAttribute» вказати значення атрибуту, через яке на елемент даного типу будуть посилатися інші елементи. |
| Альтернативний шлях виконання | 1. Якщо атрибут, до якого буде проведено посилання, вже має обчислюваний атрибут, за яким елемент-стартова точка посилання може побудувати зв'язок, кроки 6-7 опускаються. |
| Виключення | - |

Таблиця 3.5 – Опис прецеденту №5

| | |
|---|---|
| Назва | Налаштування посилань елемента заміни |
| ID | 5 |
| Опис | Адміністратор бажає налаштувати елемент-заміну для певного типу елементів |
| Актори | Адміністратор системи |
| Організаційні переваги | Типи, для яких існує налаштування елементу-заміни, будуть приховані на результуючому графі, а посилання на них будуть перенесені на елемент-заміну |
| Частота використання | 15% |
| Причина виникнення | Адміністратору необхідно налаштувати приховування певних типів елементів на графі мережевої інфраструктури |
| Передумова | Адміністратор обрав необхідні типи елементу графа для налаштування посилання на елемент-заміну |
| Постумова | Конфігурація елементу-заміни мережі має обов’язково включати в себе значення типу посилання, за яким буде визначатися елемент-заміна. |
| Головний шлях виконання | 1. Користувач повинен перейти на сторінку типу, з якого буде спрямоване висхідне посилання, згідно кроків, описаних у прецеденті №1;<br>2. При потраплянні на сторінку налаштування типу перейти на вкладку «General»; |

Продовження таблиці 3.5

| | |
|---|---|
| Головний шлях виконання | 3. Перевести важіль групи «Replace Link» у ввімкнений стан;<br>4. В групі «Replace Link» встановити для поля «Type» значення типу посилання для проведення заміни та встановити значення важеля «IsDirect» в залежності від напряму посилання. |
| Альтернативний шлях виконання | 1. Для видалення посилання-заміни необхідно перевести важіль групи «Replace Link» у вимкнений стан. |
| Виключення | - |

Таблиця 3.6 – Опис прецеденту №6

| | |
|---|---|
| Назва | Налаштування посилань елемента на групу |
| ID | 6 |
| Опис | Адміністратор бажає налаштувати посилання на групу для певного типу елементів мережевої інфраструктури |
| Актори | Адміністратор системи |
| Організаційні переваги | Типи, для яких існує налаштування посилання на групу, будуть представленні у вигляді ієрархії груп елементів |
| Частота використання | 20% |
| Причина виникнення | Адміністратору необхідно налаштувати посилання на групу для необхідного типу елемента графа |

Продовження таблиці 3.6

| | |
|---|---|
| Передумова | Адміністратор обрав необхідний типи елементу графа для налаштування групи |
| Постумова | Конфігурація посилання на групу має обов’язково включати в себе значення типу посилання, за яким буде визначатися елемент-заміна. |
| Головний шлях виконання | 1. Користувач повинен перейти на сторінку типу, з якого буде спрямоване висхідне посилання, згідно кроків, описаних у прецеденті №1;<br>2. При потраплянні на сторінку налаштування типу перейти на вкладку «General»;<br>3. Перевести важіль групи «Group Link» у ввімкнений стан;<br>4. В групі «Group Link» встановити для поля «Type» значення типу посилання га групу та встановити значення важеля «IsDirect» в залежності від напряму посилання. |
| Альтернативний шлях виконання | 1. Для видалення посилання на групу необхідно перевести важіль групи «Group Link» у вимкнений стан. |
| Виключення | - |

Таблиця 3.7 – Опис прецеденту №7

| | |
|---|---|
| Назва | Перегляд графу мережі для сервісу |
| ID | 7 |

Продовження таблиці 3.7

| | |
|---|---|
| Опис | Користувач бажає переглянути граф мережевої інфраструктури для сервісу |
| Актори | Користувач |
| Організаційні переваги | Користувач може переглянути граф мережі для обраного сервісу |
| Частота використання | 100% |
| Причина виникнення | Користувачу необхідно досліди стан мережевої інфраструктури для певного сервісу |
| Передумова | Користувач перейшов на головну сторінку клієнта nvt-fe |
| Постумова | - |
| Головний шлях виконання | 1. Користувач повинен заповнити поле «Instance Name» з назвою екземпляру сервісу;<br>2. При завантаженні графу користувач може приступити до огляду графу. |
| Альтернативний шлях виконання | 1. Для пошуку необхідного сервісу серед списку сервісів користувач може скористатися меню «Filters» для фільтрації результатів за визначеними критеріями;<br>2. Користувач може налаштувати кількість відображуваних на одній сторінці сервісів через меню пагінації в правій нижній частині панелі «Services»; |

Продовження таблиці 3.7

| | |
|---|---|
| Альтернативний шлях виконання | 3. Користувач може здійснювати переходи між сторінками списку сервісів з використанням меню переходу в лівій нижній частині панелі «Services». |
| Виключення | - |

Таблиця 3.8 – Опис прецеденту №8

| | |
|---|---|
| Назва | Перегляд властивостей елементу графу |
| ID | 8 |
| Опис | Користувач бажає переглянути властивості одного з елементів, присутніх на графі |
| Актори | Користувач |
| Організаційні переваги | Користувач може отримати інформацію про певний елемент на графі |
| Частота використання | 25% |
| Причина виникнення | Користувачу необхідно визначити властивості одного з присутніх на графі елементів |
| Передумова | Користувач обрав сервіс, граф якого він бажає переглянути |
| Постумова | - |
| Головний шлях виконання | 1. Користувач повинен обрати елемент, який він бажає переглянути;<br>2. Після появи панелі інформації про елемент мережі, значення атрибутів елементу можна переглянути на вкладці «Summary»; |

Продовження таблиці 3.8

| | |
|---|---|
| Головний шлях виконання | 3. На вкладці «Alarms» користувач може отримати інформації про проблеми в роботі елемента. |
| Альтернативний шлях виконання | 1. Для пошуку необхідного сервісу серед списку сервісів користувач може скористатися меню «Filters» для фільтрації результатів за визначеними критеріями;<br>2. Користувач може налаштувати кількість відображуваних на одній сторінці сервісів через меню пагінації в правій нижній частині панелі «Services»;<br>3. Користувач може здійснювати переходи між сторінками списку сервісів з використанням меню переходу в лівій нижній частині панелі «Services». |
| Виключення | - |

Таблиця 3.9 – Опис прецеденту №9

| | |
|---|---|
| Назва | Спостереження за змінами графу |
| ID | 9 |
| Опис | Користувач бажає переглядати зміни в графі обраного сервісу в режимі реального часу |
| Актори | Користувач |
| Організаційні переваги | Користувач спостерігає за графом сервісу в режимі реального часу |
| Частота використання | 25% |

Продовження таблиці 3.9

| | |
|---|---|
| Причина виникнення | Користувачу необхідно виконувати спостереження за змінами мережевої інфраструктури певного сервісу |
| Передумова | Користувач обрав сервіс, граф якого він бажає переглянути |
| Постумова | - |
| Головний шлях виконання | 1. Користувач очікує на появу змін в стані зовнішньої системи;<br>2. Після обробки системою повідомлень про змін користувач може переглянути актуальний стан графу, який буде автоматично оновлено. |
| Альтернативний шлях виконання | - |
| Виключення | - |

Таблиця 3.10 – Опис прецеденту №10

| | |
|---|---|
| Назва | Спостереження за станом елементів графу |
| ID | 10 |
| Опис | Користувач бажає дізнатися, чи існують проблеми в роботі мережевої інфраструктури, і якщо так, в яких саме її елементах |
| Актори | Користувач |
| Організаційні переваги | Користувач отримує інформацію про стан мережевої інфраструктури |
| Частота використання | 25% |

Продовження таблиці 3.10

| | |
|---|---|
| Причина виникнення | Користувачу необхідно отримати інформацію про роботу елементів мережевої інфраструктури |
| Передумова | Користувач обрав сервіс, граф якого він бажає переглянути |
| Постумова | - |
| Головний шлях виконання | 1. Користувач переглядає елементи мережі, що працюють в звичному режимі роботи (позначені зеленим кольором);<br>2. Користувач переглядає елементи мережі, що працюють з незначними проблемами (позначені жовтим кольором);<br>3. Користувач перегладає елементи мережі, що працюють зі значними проблемами (позначені червоним кольором). |
| Альтернативний шлях виконання | 1. Інформацію про кількість проблем елемента користувач може побачити на піктограмі, зображену в лівій верхній частині моделі елемента. |
| Виключення | - |

Таблиця 3.11 – Опис прецеденту №11

| | |
|---|---|
| Назва | Перегляд зв’язків між елементами графа |
| ID | 11 |

Продовження таблиці 3.11

| | |
|---|---|
| Опис | Користувач бажає дізнатися, які існують зв’язки між елементами графа |
| Актори | Користувач |
| Організаційні переваги | Користувач отримує інформацію зв’язки в мережевій інфраструктурі сервісу |
| Частота використання | 25% |
| Причина виникнення | Користувачу необхідно отримати інформацію про наявні зв’язки в графі сервісу |
| Передумова | Користувач обрав сервіс, граф якого він бажає переглянути |
| Постумова | - |
| Головний шлях виконання | 1. Користувач обирає на графі зв’язок, інформацію про який він хоче отримати;<br>2. Інформація про параметри зв’язку користувач переглядає на панелі інформації про зв’язок, що з’являється після обрання необхідного зв’язку; |
| Альтернативний шлях виконання | - |
| Виключення | - |

## 3.2 Висновки до розділу

В даному розділі були встановлені основні ролі користувачів, кожен з яких відповідає за взаємодії з різними частинами системи. Визначено основні сценарії використання системи, кожен з яких було детально описано з вказуванням його учасників, кроків виконання. Проведення даного опису дозволить визначити усі варіанти поведінки системи, які необхідно враховувати під час реалізації системи.

# 4 СТРУКТУРНА СХЕМА СИСТЕМИ

## 4.1 Загальна інформація про структурну схему системи

Структурна схема система описує компоненти, з яких складається система, та зв’язки між ними. При проектуванні структурної схеми системи nvt було вирішено розподілити компоненти системи на відокремлені модулі, кожен з яких представляє собою одиницю функціоналу системи (сервіс, БД, бібліотека тощо). В результаті, було виділено 14 окремих компонентів, кожен з яких відповідає за певну частину функціоналу системи. Структурна схема система представлена у додатку Б.

В залежності від функціоналу, які виконує компонент, їх можна розбити на наступні групи:

- компоненти, що відповідають за отримання даних з висхідних систем та формування графа;
- компоненти, які беруть участь в процесі відображення побудованого графа;
- компоненти, завданням яких є управління конфігурацією.

## 4.2 Опис компонентів для побудови графа мережі

Частина системи, завданням якої є отримання подій з зовнішніх систем та побудова графа складається з наступних компонентів: streaming-core, rabbitmq, nvt-composer, nvt-pg-graph-streamer, nvt-pg-graph-streamer-db, nvt-service-streamer, nvt-service-streamer-db.

Компонент streaming-core є фундаментом, на основі якого будуються всі сервіси-стрімери. Він представляє собою бібліотеку, написану на мові програмування Go, яка надає користувачам API для опису коннектора. Даний API включає в себе визначення основних понять сервісу-стрімера, таких як коннектор, менеджер та конструктор. Також дана бібліотека включає в себе модель подій, якими сервіси-стрімери обмінюються з сервісом nvt-composer, який використовує їх

для оновлення графа. І нарешті, до складу бібліотеки також входить інтерфейс та реалізація висхідного коннектора для RabbitMQ. Даний компонент надає інтерфейс для компонентів nvt-pg-graph-streamer та nvt-service-streamer.

Компонент rabbitmq представляє собою розгорнуту систему обміну повідомлень RabbitMQ, яка виступає посередником між сервісами-стрімерами та сервісом nvt-composer. Даний компонент використовується сервісами nvt-pg-graph-streamer та nvt-service streamer для надсилання подій оновлення висхідних систем, та сервісом nvt-composer для обробки надісланих повідомлень та оновлення графа відповідно до вказаних змін.

Головним компонентом, що виконує побудову графу є nvt-composer. Даний компонент представляє собою мікросервіс, написаний на мові програмування Go. Основним завданням сервісу є зчитування надісланих в RabbitMQ повідомлень та їх обробка, що може включати в себе наступні пункти:

- обробка та додавання вузлів графу;
- обраховування значень для обчислюваних атрибутів в конфігурації;
- визначення посилань для елементів-замінників та елементів, що належать до певної групи;
- створення посилань між елементами за конфігурацією обчислюваних атрибутів елемента мережі;
- обраховування посилань між елементами за вказаним в конфігурації шляхом обходу графу;
- обробка інформації з систем моніторингу про виникнення проблем елемента мережі (alarms);
- проведення процедури синхронізації для відновлення консистентності даних між даними в системі nvt та висхідними системами.

Компонент nvt-composer використовує компонент nvt-framework для отримання конфігурації елементів мережі, на основі якої буде здійснюватися їх обробка. Для зчитування надісланих повідомлень та публікації повідомлень про необхідність оновлення вузлів nvt-composer використовує компонент rabbitmq. І нарешті, для збереження оброблених даних використовується компонент nvt-db.

Для демонстрації роботи системи було розроблено два тестових стрімери, які оперують з набором тестових даних. Першим з них є компонент nvt-pg-graph-streamer. nvt-pg-graph-streamer є демонстраційним мікросервісом-стрімером, написаним на мові програмування Go. Його завданням є отримання даних з висхідної системи, яка в даному випадку представлена компонентом nvt-pg-graph-streamer-db, що представляє собою базу даних PostgreSQL, в якій дані про елементи мережі зберігаються на основі моделі EAV. Для отримання даних з даної БД стрімер виконує прослуховування на заздалегідь узгодженому каналі, в який через механізм тригерів БД повідомляє про зміни в висхідній системі.

Компоненти nvt-service-streamer та nvt-service-streamer-db відповідають за другу демонстраційну висхідну систему. На відміну від nvt-pg-graph-streamer, тут містяться не самі елементи мережі, а елементи – екземпляри сервісів, які вказуються які з елементів мережі використовується для надання певних послуг. Дані елементи потім будуть використані сервісом nvt-be в якості стартової точки для побудови графу. Дані про сервіси містяться в базі даних, за яку відповідає компонент nvt-service-streamer-db.

Всі сервіси-стрімера використовують компонент streaming-core для реалізації наданого ним інтерфейсів коннектора та конструктора, компонент nvt-framework для отримання конфігурації елементів графа.

## 4.3 Опис компонентів для відображення графу

До набору компонентів, які забезпечують відображення графу, входять nvt-be, nvt-db, nvt-fe.

Компонент nvt-be представляє собою мікросервіс, написаний на мові програмування Go. Він представляє API для отримання графу згідно ідентифікатору сервіса та для підписки на оновлення графа.

Для своєї роботи nvt-be взаємодіє з компонентом nvt-db для отримання графу мережевої інфраструктури та компонентом nvt-framework для отримання конфігурації, а саме використання користувацьких запитів, якщо такі було вказано.

Компонент nvt-fe відповідає за відображення користувачу графу мережевої інфраструктури згідно обраного мережевого сервісу. Його влаштування можна деталізувати на два підкомпоненти – Vuex сховище, яке виступає посередником між серверною частиною та моделлю користувацького інтерфейсу.

## 4.4 Опис компонентів для управління конфігурацією системи

Конфігурація системи використовується в усіх сервісах-стрімерах та у nvt-composer. Для спрощення взаємодії з сервісом конфігурації було розроблено компонент nvt-framework, який представляє собою бібліотеку на мові Go, що включає набір рішень, спільних для усіх сервісів в системі. Дана бібліотека представляє зручний інтерфейс, які інші користувач можуть використати для отримання конфігурації системи.

Для своєї роботи nvt-framework потребує компонент nvt-config-be, який і виступає постачальником конфігурації. Сервіс написано на мові Go, для збереження конфігурації використовує компонент nvt-config-db.

Nvt-config-db представляє собою розгорнуту БД PostgreSQL, в якій зберігається конфігурація системи. В якості моделі даних використовується моделі EAV.

Графічний інтерфейс для управління конфігурацією надає сервіс nvt-config-fe. Даний сервіс написано з використанням архітектурного патерну MVVM. В даній реалізації патерну в якості моделі виступає сховище Vuex, яке містить у собі всю логіку взаємодії з сервісом конфігурації, що дозволяє залишити логіку представлення чистою та простою для розуміння.

## 4.5 Висновки до розділу

В даному розділі було описано влаштування структурної схеми для розроблюваної системи. Компоненти згруповано в модулі залежно від функції, яку вони повинні виконувати.

Одним з основних модулів містить групу компонентів, що забезпечують отримання даних з висхідних систем. Основними компонентами даної групи є сервіси-стрімери, які здійснюють отримання інформації про зміни стану систем, та сервіс nvt-composer, який зчитує дані надіслані повідомлення про зміни стану та перебудовує оновлює граф згідно вказаної інформації. Для спрощення та уніфікації написання сервісів-стрімерів використовується компонент streaming-core, який містить у собі інтерфейс який повинен реалізувати коннектор та модель подій.

Модуль, що відповідає за відображення графу, представляє собою набір сервісів, що використовують побудований сервісом nvt-composer граф для його відображення через графічний інтерфейс.

Остання група представляє собою набір компонентів, які відповідають за роботу з конфігурацією системи. Сама конфігурація міститься в реляційній БД з використанням моделі даних EAV. Доступ до неї компоненти отримують через сервіс nvt-config-be. Для зміни конфігурації передбачено графічний інтерфейс, за який відповідає компонент nvt-config-fe.

В результаті був досягнутий необхідний рівень гнучкості архітектури, який дозволяє задовольнити поставленим перед системою вимогам.

# 5 ВИБІР ТА ОБГРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ

## 5.1 Вибір мови програмування для серверної частини системи

В архітектурі клієнт-сервер сервер грає роль обробника запитів на вимогу клієнтів. Сервер передбачає надавання послуг безлічі користувачів, а тому може бути перевантаженим надмірною кількістю запитів. Головною вимогою, яка зазвичай висуваються до сервера є пропускна спроможність. В контексті мережевих застосунків даний термін представляє метрику того, скільки запитів сервер може обробити за визначений інтервал часу.

Golang, також відома як Go, є мовою програмування, розроблена Google Робом Пайком, Робертом Грізмером та Кеном Томпсоном. Основні особливості мови включають: С-подібний синтаксис, сильна статична типізація, структурна типізація, безпечна робота з пам’яттю за допомогою збирача сміття (GC) та вбудована підтримка багатониткове програмування на основі принципів CSP.

Оскільки для системи необхідно забезпечити швидку передачу інформації про зміни в під’єднаних системах, необхідно мінімізувати час, що проходить від впровадження зміни в висхідній системі до його обробки та її наступного відображення. Golang представляє собою ідеальний баланс між швидкістю виконання написаних програм та використанням пам’яті. Згідно результатів проведеного в 2018 році порівняння основних мов програмування було встановлено, що потреби Go в плані використання пам’яті є значно меншими за свого найближчого конкурента серед мов програмування з автоматичним управлінням пам’яттю а саму мову було визнано оптимальним рішенням для розробки великих розподілених систем[6].

Однією з найбільш важливих характеристик для розподілених мережевих застосунків є пропускна здатність системи. В системах даного типу більшу частину часу займають задачі, пов’язані з введенням-виведенням інформації (запити в базу даних, прийом запитів, надсилання відповідей), необхідна максимальна утилізація процесорних ресурсів. Мова програмування Golang використовує модель

багатонитковості на основі горутин та каналів, які в свою чергу є реалізацією моделі CSP, що була розроблена Тоні Хоаром у 1978р. Дана модель суттєво спрощує написання розподілених систем, та дозволяє уникнути багатьох проблем, що виникають у традиційній моделі на основі системних потоків та механізмів блокування[7].

З недоліків мови Go можна виділити обмежені можливості мови для узагальненого програмування. Для вирішення даної проблеми мова надає механізм порожнього інтерфейсу, що дозволяє змінним приймати значення будь-якого типу. В свою чергу, це може призвести до появи помилок типізації під час роботи системи, оскільки компілятор не може встановити правильність типів на етапі компіляції коду.

Мова програмування Java є об’єктно-орієнтованою мовою програмування, що була створена компанією «Sun Microsystems» у 1995 році. Серед основних особливостей мови можна виділити наступні: простий та зрозумілий синтаксис, незалежність від середовища виконання, висока продуктивність виконання, інтерпретований характер мови з можливістю динамічного зв’язування модулів.

Головною перевагою мови Java є кроссплатформеність. Програма, написана на мові Java, може працювати на будь-якій платформі, для якої існує реалізації віртуальної машини Java, без змін у початковому коді або повторної компіляції. Дана особливість досягається за рахунок використання байт-коду, проміжної форми, у яку компілюється код програми. Потім даний байт-код оброблюється віртуальною машиною Java, яка перетворить його на пристосований до конкретного середовища виконання код.

Також до переваг мови можна занести наявність розвинутої екосистеми. Java внаслідок свого великого рівня використання представляє широкий набір рішень для вирішення основних проблем, що можуть виникнути під час розробки розподілених мережевих систем.

Серед недоліків мови можна виділити використання пам’яті. Оскільки мова передбачає додатковий етап компіляції у байткод, при виконанні програми

віртуальній машині необхідно скомпілювати байткод у код системи, в якій виконується програма.

В результаті проведеного аналізу було Go в якості мови серверної частини. Порівняно з мовою Java, Go компілюється одразу для певного середовища виконання, а тому значно ефективніше використовує системні ресурси. Хоча екосистема даної мови є значно меншою, наявність широкої стандартної бібліотеки та досить високий рівень існуючих бібліотек стали ключовими факторами при прийнятті рішення.

## 5.2 Вибір бази даних для збереження даних про стан мережі

Мережеву інфраструктуру можна представити у вигляді набору пов'язаних між собою вузлів. Дана модель оптимально підходить під теорію графів. Саме тому в якості бази даних для збереження інформації про стан інфраструктури було вирішено обрати графову базу даних.

Neo4J є одним з найвідоміших представників серед графових баз даних. Дана БД зосереджена повністю на представленні вузлів та зв’язків між ними. Головною особливістю Neo4J є запровадження зручного механізму для взаємодії з даними графу через мову запитів Cypher. Cypher дозволяє здійснювати основні операції над вузлами та зв’язками між ними, описувати шляхи, за якими необхідно здійснювати пошук вузлів, та інші операції, пов’язані з роботою з графом.

Хоча Neo4J надає широкий набір можливостей для роботи з пов’язаними даними, графова модель є єдиною підтримуваною на даний момент. Якщо система потребує використання декількох моделей даних, для їх зберігання необхідно використовувати окрему базу даних.

На відміну від баз даних, що фокусуються виключно на роботі з графами, ArangoDB є багатомодельним рішенням та окрім можливостей графової БД може використовуватися як документоорієнтована БД та сховище типу «ключ-значення». Попри те, що структура мереж відповідає графовій моделі, не всі дані про інфраструктуру є ієрархічними. Наприклад, інформація по метрикам та проблемам в

мережі, яка зазвичай міститься в окремій системі, не має ієрархічного спрямування та прив’язана до конкретної сутності. Для такого типу інформації, використовується документноорієнтована модель БД.

В ArangoDB робота з базою даних здійснюється через мову запитів AQL, що підтримує роботу з усіма моделями збереження даних, спрощуючи роботи з запитами, в яких необхідні даних з різних моделей[8].

Оскільки розроблювана система потребує використання декількох моделей даних, ArangoDB було прийнято в якості рішення для збереження даних про стан графу мережевої інфраструктури.

## 5.3 Вибір бази даних для збереження конфігурації системи

Модель конфігурації має ієрархічний характер. На перший погляд, для збереження конфігурації системи можна за аналогією також використати графову базу даних з орієнтованим графом, в якому ребра будуть вказувати напрямок ієрархії. Проте, на відміну від моделі мережі, в якій вузли є самостійними цілісними частинами, центральним елементом в моделі конфігурації системи є сутності-об’єктні типи, що складаються з атрибутів. Об’єктні типи є ієрархічними сутностями, у яких атрибути батьківського об’єктного типа «наслідуються» об’єктними типами-нащадками. Тобто, самі атрибути є відносно самостійною частиною, які пов’язані не лише  напряму з об’єктним типом, а також з усіма його нащадками.

MySQL – вільна відкрита система керування реляційними даними, що є багатофункціональною системою і підтримує більшість функціональних можливостей SQL прямо чи опосередковано. MySQL є досить гнучким рішенням, що робить дану БД популярним вибором при розробці веб-застосунків. Додаткові функції включають в себе наявність незалежність платформи, відповідність ACID та простота підтримки.

Одним з недоліків MySQL є неповна підтримка стандарту SQ, що зменшує загальну сумісність запитів з іншими БД. Також MySQL надає малий набір засобів

за спостереження за змінами стану даних в БД, обмежуючись лише стандартною для більшості БД фізичною реплікацією через WAL-лог.

PostgreSQL є відкритою реляційною СУБД, перелік особливостей якої включає в себе високий рівень підтримки стандарту SQL, високі можливості з розширення базового функціоналу та велику кількість сторонніх бібліотек, що надають додатковий функціонал.

Однією з важливих переваг PostgreSQL є широкий набір підтримуваних типів даних. PostgreSQL підтримує масиви даних для збереження змінного набору даних певного типу. Також надається підтримка неструктурованих даних через формат JSON.

В якості рішення для збереження конфігурації системи було обрано реляційну модель даних, оскільки вона дозволяє задовольнити умову ієрархічності конфігурації, при цьому дозволяючи встановити обмеження цілісності даних. У якості реляційної бази даних було обрано PostgreSQL внаслідок широкого набору підтримуваних типів, оскільки модель конфігурації розроблюваної системи передбачає використання неструктурованих даних для налаштування адаптерів атрибутів.

## 5.4 Вибір системи обміну повідомленнями

Для взаємодії між сервісами, що відповідають за отримання даних с висхідних систем, та сервісів, що відповідають за їх оброку, необхідно мати «посередника», що буде зберігати повідомлення, та дозволить запобігти перезавантаження сервісу, що обробляє данні. Для виконання цієї ролі використовується система обміну повідомленнями.

RabbitMQ є найбільш використовуваним рішенням серед систем обміну повідомленнями. RabbitMQ представляє собою систему обміну повідомленнями, написану на мові Erlang. Система підтримує управління рівнем доступу користувачів, шифрування даних, зберігання повідомлень на жорсткий диск у

випаду запланованого періоду перебоїв активності, роботу в кластері та дуплікацію запущених сервісів.

Основною особливістю RabbitMQ є її централізована архітектура. RabbitMQ представляє собою брокер для обміну повідомленнями, що реалізує протокол AMQP низького рівня та модель виробника-споживача. Він є посередником між двома додатками, коли в процедурі обробки зв'язку.

ZeroMQ є розподіленою бібліотекою для побудови розподілених систем обміну повідомленнями. На відміну від RabbitMQ, дане рішення не є готовим сервісом, який використовують інші сервіси. Натомість, ZeroMQ представляє контракт, який повинні реалізувати на його основі основний функціонал для обміну повідомленнями.

В якості системи обміну повідомлення було обрано RabbitMQ. ЇЇ архітектурний підхід на основі центрального вузла-брокера відповідає архітерному підходу системи, в якій існує один центральний вузол – сервіс, що обробляє дані та формує граф, та безліч сервісів, що надсилають дані.

## 5.5 Вибір технологій для проектування клієнтської частини

В якості мови програмування клієнтської частини найбільш використовуваними рішеннями є мови програмування TypeScript та JavaScript.

JavaScript є динамічною мовою програмування з підтримкою об’єктно-орієнтованого програмування на основі прототипів та можливостей функціонального програмування, що дозволяє розробнику використовувати найбільш підходящу для системи парадигму або навіть поєднувати декілька парадигм.

Головним недоліком мови є динамічна слабка типізація. На початкових етапах це дозволяє швидко проектувати прототипи системи, проте з часом виникають складності з підтримкою таких систем і розробникам доводиться звертатися до сторонніх рішень для забезпечення типізації або написанню модульних тестів для перевірки типів вхідних аргументів.

Мову програмування було представлено Microsoft у 2012 році, та позиціонується як засіб для розширення можливостей мови JavaScript. Головним призначенням мови є додавання до звичного синтаксису JavaScript статичної типізації, що дозволяє на етапі компіляції запобігти цілому класу помилок, пов’язаних з неправильним використанням типів даних.

Основною перевагою мови, TypeScript є її зворотна сумісність з мовою JavaScript. Це дозволяє напряму використовувати великий обсяг вже існуючих бібліотек. Ще одним критерієм для вибору даної мови є підвищена продуктивність розробника. Наявність типізації дозволяє не лише запобігати помилкам, пов’язаним з хибним використанням типів, але й полегшити роботу з кодовою базою в цілому (рефакторинг, пошук елементів по імені, інтеграція з безліччю існуючих текстових редакторів та IDE).

Оскільки мова TypeScript є строгою надмножиною мови JavaScript з доданою статичною типізацію, цю мову було обрано для розробки клієнтської частини системи.

Хоча розробка користувацького інтерфейсу може бути здійснена з використанням вбудованих можливостей мови JavaScript, використання фреймворку дозволяє збільшити продуктивність розробника та зменшити час на розробку рішення для основних проблем, що виникають при проектуванні клієнтської системи.

Фреймворк Angular є наступником фреймворку AngularJS, який було розроблено з метою переосмислення отриманого досвіду під час розробки AngularJS.

Найбільшою перевагою Angular є широкий набір вбудованого функціонала. Angular представляє собою повноцінний фреймворк, який включає в себе рішення для всіх можливих проблем, що виникають при розробці клієнтських застосунків.

Також до переваг Angular входить наявність засобу командного рядка. Це дозволяє автоматизувати виконання повторюваних при розробці процедур, таких як генерація скелету застосунку, створення компонентів, додавання сторонніх залежностей тощо.

Для розробки інтерфейсу користувача було використано фреймворк VueJS. Даний фреймворк є представником компонентного підходу до розробки графічних інтерфейсів, в якому клієнтська частина представляє собою композицію компонентів, кожен з яких відповідає за певну частину інтерфейсу. Такий підхід сприяє повторному використанню елементів інтерфейсу та дозволяє спростити побудову консистентних інтерфейсів.

Також порівняно з бібліотеками, що оперують з вузлами DOM напряму, дана бібліотека використовує декларативний підхід, згідно якого описується за допомогою шаблонів вигляд системи в залежності від вхідних даних що отримав компонент (props) та його внутрішнього стану (data), а власне модифікацію DOM бере на себе фреймворк.

VueJS, за словами авторів, є «прогресивним фреймворком»[9]. Це означає, що фреймворк розподілений на модулі, які користувач може використовувати в залежності від власних потреб. При розробці представленої у даній дисертації системі окрім головного модуля було також використано модуль, що відповідає за управління станом (Vuex), та модуль для здійснення роутингу на клієнтській частині (vue-router).

В результаті порівняння було обрано фреймворк VueJS для розробки клієнтської частини. Даний фреймворк надає можливість поступово додавати модулі за необхідністю, та представляє більш просту модель компонентів.

## 5.6 Вибір засобів для забезпечення взаємодії в реальному часі

Для забезпечення відображення оновлень графу користувацьким інтерфейсом необхідно забезпечити взаємодію між клієнтом та сервером в реальному часі.

Server Sent Events (SSE) представляє собою технологію для передачі сервером автоматичних оновлень клієнту через HTTP-з’єднання. API даної технології є стандартизованою частиною специфікації HTML5. SSE описує, як сервери можуть ініціювати передачу даних після встановлення початкового з'єднання з клієнтом. Областю застосування данної технології є надсилання оновлень повідомлень або

безперервних потоків даних клієнту. Головним недоліком даної технології є обмеження кількості з’єднань в обсягу від 4 до 10 в залежності від браузера[10].

WebSocket – протокол для забезпечення дуплексного каналу обміну інформації через TCP-з’єднання. Протокол було стандартизовано IETF стандартом RFC6455 у 2011 році. Протокол WebSocket дозволяє забезпечити взаємодію між клієнтом та веб-сервером полегшуючи передачу даних у реальному часі з сервера. Це стає можливим завдяки забезпеченню способу надсилання контенту сервером без попереднього запиту клієнта, а також передачі повідомлень в обидві сторони, зберігаючи з'єднання відкритим. Таким чином, між клієнтом і сервером може відбуватися двостороння бесіда.

Протокол WebSocket було встановлено найбільш підходящим рішенням для системи внаслідок наявного в технології SSE обмеження на кількість відкритих з’єднань в межах браузера.

## 5.7 Висновки до розділу

В даному розділі було проведено аналіз обраних для розробки технологій. Реалізація система використовує мікросервісну архітектуру. Дане рішення висуває ряд вимог до складових системи: ефективне використання системних ресурсів, необхідність вирішення проблеми взаємодії складових системи, децентралізація процесу управління даними системи.

В якості мови програмування сервісної частини була обрана мова програмування Go. Використання даної мови має наступний ряд переваг: низькі показники використання пам’яті порівняно з конкурентами, першокласна підтримка засобів багатониткового програмування, широка стандартна бібліотека .

Для відображення даних про мережу була вибрана графова база даних ArangoDB. Головними перевагами даної БД перед конкурентами є підтримка мультимодельного збереження даних залежно від потреб системи.

Модель конфігурації висуває ряд вимог до цілісності даних, а тому в якості механізму зберігання даних було обрано PostgreSQL, яка окрім підтримки основних

можливостей реляційної БД також надає широкий набір засобів для представлення нереляційних даних (наприклад, підтримка зберігання документів в форматі JSON).

В якості рішення, що забезпечить контроль обміну повідомленнями між сервісами-стрімерами було обрано RabbitMQ внаслідок брокерної архітектури, що відповідає архітектурі системи (безліч надсилачів, один приймач).

Було проаналізована існуючі мови програмування, що використовуються для побудови клієнтських інтерфейсів. В результаті дослідження було вирішено обрати мову TypeScript. Наявність статичної типізації дозволяє усунути на етапі компіляції цілий ряд проблем, пов’язаних з неправильним використанням типів. Також вагомим аргументом на користь використання даної мови є легкість інтеграції з існуючою екосистемою.

При визначенні технологій розробки клієнтської частини було вирішено використати фреймворк VueJS. Даний фреймворк дозволяє декларативно описати інтерфейс користувача на основі компонентної моделі, що сприяє декомпозиції клієнтської частини на окремі частини, значно полегшуючи розробку.

Для оновлення графу було проведено порівняльний аналіз механізмів забезпечення двостороннього зв’язку між клієнтом та сервером. В ході даного дослідження технологію було обрано протокол WebSocket внаслідок відсутності обмежень щодо кількості відкритих з’єднань та високого рівня підтримки сучасними браузерами.

# 6 ER-ДІАГРАМА

Для позначення обмежень таблиць використовуються наступні скорочення:

- PK – Primary Key (первісний ключ);
- GEN – Generated (значення колонки генерується автоматично);
- FK – Foreign Key (зовнішній ключ);
- NN – Not Null (обов’язкова колонка);
- UN – Unique (унікальне значення).

## 6.1 Схема БД для сервісу nvt-config

ER-діаграма сервісу представлена у додатку В.

Таблиця 6.1 – Опис таблиці object_types

| Назва поля | Тип | Обмеження |
|---|---|---|
| object_type_id | UUID | PK, GEN |
| name | VARCHAR(64) | |
| system_name | VARCHAR(256) | UNIQUE |
| parent_id | UUID | FK object_types(object_type_id) |
| element_type | VARCHAR(30) | |
| circuit_path | TEXT | |
| query | TEXT | |
| group_link | VARCHAR(100) | |
| replace_link | VARCHAR(100) | |

Таблиця 6.2 – Опис таблиці attribute_types

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_type_id | SMALLINT | PK |

Продовження таблиці 6.2

| Назва поля | Тип | Обмеження |
|---|---|---|
| name | VARCHAR(30) | NN |

Таблиця 6.3 – Опис таблиці attr_adapters

| Назва поля | Тип | Обмеження |
|---|---|---|
| adapter_type | VARCHAR(100) | PK |
| fields | VARCHAR(64)[] | |

Таблиця 6.4 – Опис таблиці attributes

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_id | UUID | PK, GEN |
| name | VARCHAR(64) | NN |
| system_name | VARCHAR(256) | UNIQUE |
| attr_type_id | SMALLINT | FK object_types(object_type_id) |
| is_calculable | BOOLEAN | |
| adapter_type | VARCHAR(100) | FK attr_adapters(attr_type_id) |
| adapter_config | JSON | |

6.2 Схема БД для сервісу nvt-pg-graph-streamer

ER-діаграма сервісу представлена у додатку Г.

Таблиця 6.5 – Опис таблиці attr_types

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_type_id | SMALLINT | PK |
| name | VARCHAR(64) | |

Таблиця 6.6 – Опис таблиці lists

| Назва поля | Тип | Обмеження |
|---|---|---|
| list_id | UUID | PK GEN |
| name | VARCHAR(64) | |
| description | TEXT | |

Таблиця 6.7 – Опис таблиці object_types

| Назва поля | Тип | Обмеження |
|---|---|---|
| object_type_id | UUID | PK GEN |
| name | VARCHAR(64) | |
| description | VARCHAR(256) | |
| parent_id | UUID | FK object_types(object_type_id) |
| is_class | BOOLEAN | |
| system_name | VARCHAR(256) | UNIQUE |

Таблиця 6.8 – Опис таблиці attributes

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_id | UUID | PK GEN |
| name | VARCHAR(64) | |
| description | TEXT | |
| is_multiple | BOOLEAN | |
| is_calculable | BOOLEAN | |
| attr_type_id | SMALLINT | FK attr_types(attr_type_id) |
| list_id | UUID | FK lists (list_id) |
| system_name | VARCHAR(256) | UN |

Таблиця 6.9 – Опис таблиці attr_object_types

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_id | UUID | FK attributes(attr_id) |

Продовження таблиці 6.9

| Назва поля | Тип | Обмеження |
|---|---|---|
| object_type_id | UUID | FK object_types(object_type_id) |
| default_value | VARCHAR(64) | |
| is_required | BOOLEAN | DEFAULT FALSE |

Таблиця 6.10 – Опис таблиці objects

| Назва поля | Тип | Обмеження |
|---|---|---|
| object_id | UUID | PK GEN |
| name | VARCHAR(64) | |
| description | TEXT | |
| object_type_id | UUID | FK object_types(object_type_id) |
| object_class_id | UUID | FK object_types(object_type_id) |
| parent_id | UUID | FK object_types(object_type_id) |
| project_id | UUID | FK object_types(object_type_id) |
| created_by | VARCHAR(64) | |
| created_when | BIGINT | GEN |
| modified_when | BIGINT | GEN |
| order_number | SERIAL | |

Таблиця 6.11 – Опис таблиці list_values

| Назва поля | Тип | Обмеження |
|---|---|---|
| list_value_id | UUID | PK GEN |
| list_id | UUID | FK lists(list_id) |
| value | VARCHAR(50) | |
| color | VARCHAR(20) | |
| system_name | VARCHAR(256) | |

Таблиця 6.12 – Опис таблиці params

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_id | UUID | FK attributes(attr_id) |
| object_id | UUID | FK objects(object_id) |
| value | VARCHAR(256) | |
| list_value_id | UUID | FK list_values(list_value_id) |
| data | TEXT | |
| date_value | BIGINT | |

Таблиця 6.13 – Опис таблиці refs

| Назва поля | Тип | Обмеження |
|---|---|---|
| attr_id | UUID | FK attributes(attr_id) |
| object_id | UUID | FK objects(object_id) |
| reference_id | UUID | FK objects(object_id) |

Окрім таблиць, в схему БД також входять тригери, перелік яких наведено в таблиці 6.14.

Таблиця 6.14 – Опис тригерів для сервісу nvt-pg-graph-streamer

| Назва тригеру | Таблиця | Тип тригера | Опис |
|---|---|---|---|
| nvt_object_created | objects | after create | Здійснює виклик pg_notify; |
| nvt_object_updated | objects | after update | Здійснює виклик pg_notify; |
| nvt_object_deleted | objects | after delete | Здійснює виклик pg_notify; |
| nvt_param_created | params | after create | Здійснює виклик pg_notify; |

Продовження таблиці 6.14

| | | | |
|---|---|---|---|
| nvt_param_updated | params | after update | Здійснює виклик pg_notify; |
| nvt_param_deleted | params | after delete | Здійснює виклик pg_notify; |
| nvt_reference_created | references | after create | Здійснює виклик pg_notify; |
| nvt_reference_updated | references | after update | Здійснює виклик pg_notify; |
| nvt_reference_deleted | references | after delete | Здійснює виклик pg_notify; |

## 6.3 Опис схеми БД сервісу nvt-be

Дані для сервісу nvt-be зберігаються в ArangoDB, яка використовує характерний для багать ох баз даних NoSQL підхід з гнучкою схемою даних. В таблицях буде наведений очікуваний тип даних для полів відповідно до стандарту ECMA-404 про формат кодування даних JSON[11]. ER-діаграма системи представлена у додатку Д.

### 6.3.1 Таблиця nodes.

Тип таблиці – document. Опис полів таблиці наведено в таблиці 6.15.

Таблиця 6.15 – Опис таблиці nodes

| Поле | Тип поля | Індекси |
|---|---|---|
| _key | string | primary |
| name | string | |
| type | string | hash |

Продовження таблиці 6.15

| Поле | Тип поля | Індекси |
|---|---|---|
| class | string | |
| createdWhen | string | |
| updatedWhen | string | |
| sourceSystem | string | |
| processedWhen | string<br>(час у форматі RFC3339) | |
| elementType | string | |
| Поле | Тип поля | Індекси |
| groupKey | string | |
| parentKey | string | |
| replaceWith | string | |
| attributes | object | |
| alarms | string | |
| severity | string | |
| tunnel | object | |
| srcObject | string | |
| collectedAttributes | object | |

### 6.3.2 Таблиця source_links

Тип таблиці – edge. Опис полів таблиці наведено в таблиці 6.16.

Таблиця 6.16 – Опис таблиці source_links

| Поле | Тип поля | Індекси |
|---|---|---|
| _key | string | primary |
| _from | string | edge |
| _to | string | edge |

Продовження таблиці 6.16

| Поле | Тип поля | Індекси |
|---|---|---|
| Name | string | |
| type | string | hash |
| srcObject | string | |
| attributes | object | |
| sourceSystem | string | |
| processedWhen | string (час у форматі RFC3339) | |
| topology | bool | |

### 6.3.3 Таблиця group_links

Тип таблиці – edge. Опис полів таблиці наведено в таблиці 6.17.

Таблиця 6.17 – Опис таблиці group_links

| Поле | Тип поля | Індекси |
|---|---|---|
| _key | string | primary |
| _from | string | edge |
| _to | string | edge |
| name | string | |
| type | string | |
| srcObject | string | |
| attributes | object | |
| sourceSystem | string | |
| processedWhen | string<br>(час у форматі RFC3339) | |
| topology | bool | |

### 6.3.4 Таблиця replace_links

Тип таблиці – edge. Опис полів таблиці наведено в таблиці 6.18.

Таблиця 6.18 – Опис таблиці group_links

| Поле | Тип поля | Індекси |
|---|---|---|
| _key | string | primary |
| _from | string | edge |
| _to | string | edge |
| name | string | |
| type | string | hash |
| srcObject | string | |
| attributes | object | |
| sourceSystem | string | |
| processedWhen | string<br>(час у форматі RFC3339) | |
| topology | bool | |

### 6.3.5 Таблиця alarms

Тип таблиці – document. Опис полів таблиці наведено в таблиці 6.19.

Таблиця 6.19 – Опис таблиці alarms

| Поле | Тип поля | Індекси |
|---|---|---|
| _key | string | primary |
| name | string | |
| description | string | |
| managedObjectId | string | hash |

Продовження таблиці 6.19

| Поле | Тип поля | Індекси |
|---|---|---|
| severity | string | |
| ackState | string | |
| raisedTime | string | |
| changedTime | string | |
| clearedTime | string | |
| reportingTime | string | |
| sourceSystem | string | |
| processedWhen | string<br>(час у форматі RFC3339) | |

## 6.4 Опис схеми БД сервісу nvt-service-streamer

Для опису структури БД даного сервісу буде проведено опис структури вкладених документів. Опис структури вкладених документів наведено в таблицях 6.19-6.22. Опис основних таблиць сервісу наведено в таблицях 6.23-6.24. ER-діаграма сервісу представлена у додатку Е.

Таблиця 6.19 – Опис документа типу serviceModel

| Поле | Тип поля |
|---|---|
| _id | objectId |
| name | string |
| state | string |
| serviceRelationship | serviceRelationship |
| serviceCharacteristic | serviceCharacteristic |
| specification | serviceSpecification |

Таблиця 6.20 – Опис документа типу serviceCharacteristic

| Поле | Тип поля |
|---|---|
| name | string |
| value | string |

Таблиця 6.21 – Опис документа типу serviceSpecification

| Поле | Тип поля |
|---|---|
| id | string |
| name | string |
| version | string |

Таблиця 6.22 – Опис документа serviceRelationship

| Поле | Тип поля |
|---|---|
| type | string |
| serviceId | string |

Таблиця 6.23 – Опис таблиці serviceInstance

| Поле | Тип поля |
|---|---|
| _id | objectId |
| serviceType | string |
| createdWhen | timestamp |
| updatedWhen | timestamp |
| serviceModel | serviceModel |
| expectedModel | expectedModel |

Таблиця 6.24 – Опис таблиці serviceSnapshot

| Поле | Тип поля |
|---|---|
| _id | objectId |
| createdDate | timestamp |

Продовження таблиці 6.24

| Поле | Тип поля |
|---|---|
| lastModifiedDate | timestamp |
| serviceModel | serviceModel |
| serviceType | string |
| snapshotState | string |
| requestState | string |
| actualServiceState | string |
| version | number |
| priority | number |

## 6.5 Висновки до розділу

В даному розділі було описано структуру даних для всіх сервісів системи. При розробці кожного сервісу була визначена оптимальна модель даних, обмеження цілісності, яким повинні відповідати дані в системі.

Для збереження даних стану мережевої інфраструктури була обрана БД ArangoDB, оскільки модель для представлення мережі підходить під модель графових баз даних, а підтримка інших моделей даних окрім графової дозволить вирішити проблему необхідності сховища для зберігання даних про метрики та проблеми, що заважають роботі елементів мережевої інфраструктури.

Модель конфігурації системи також відповідає моделі графових баз даних, проте до атрибутів конфігурації висуваються також додаткові вимоги до цілісності даних, саме тому було обрано реляційну базу даних.

Модель сервісів, які обробляє сервіс nvt-service-streamer, представляє собою пласку модель з обмеженою кількістю зв’язків між сутностями. Саме тому в якості середовища зберігання даних було обрану базу даних MongoDB, яка оптимально підходить для зберігання документів.

# 7 РОЗРОБКА БІЗНЕС-ЛОГІКИ

## 7.1 Виділення загальних частин системи

При проектуванні розподілених програмних систем на основі мікросервісної архітектури можуть виникати спільні проблеми, що потребують рішення у декількох сервісах. Дані проблеми вирішуються шляхом виділення окремих модулів, оформлених у вигляді бібліотек, які підключає сервіс. При цьому бажано забезпечити модульність даних бібліотек, тобто сервіс повинен мати можливість підключити лише ту частину функціоналу, яка необхідна його роботі. Розглянута в дисертації система виділяє увесь спільний функціонал в дві бібліотеки – streaming-core та nvt-framework.

Бібліотека streaming-core включає в себе основні абстракції, за допомогою яких можна побудувати сервіс-стрімер. Використання даної бібліотеки не є критичною умовою при побудові сервіса-стрімера, проте є бажаним, оскільки вона надає містить опис моделі подій оновлення, а також бере на себе процес управління життєвим циклом стримера, та дозволяє натомість зосередитися розробника на вирішенні задачі отримання подій оновлення з висхідної системи. Згідно бібліотеки, сервіс-стрімер повинен складатися з наступних частин – connector, manager, builder. Діаграма класів бібліотеки представлена у додатку Ж.

Connector є центральною абстракцією системи. Даний інтерфейс представляє собою абстрацію, що є широко розповсюдженою та використовуваною при проектуванні систем стрімінгу даних. Коннектор в контексті розробки програмного забезпечення вирішує одну або декілька класів задач[12]:

- встановлення зв’язку між системами, передачі даних між компонентами систем;
- координація роботи взаємопов’язаних систем;
- перетворення даних між системами;
- полегшення формування зв’язку між системами.

Інтерфейс коннектора зображено на рис. 7.1. Він складається з наступних методів:

- StartStreaming – в даному методі міститься опис основного функціоналу коннектора. На вхід даний метод отримує канал через який можна здійснювати контроль над режимом його роботи, стартовий режим роботи та функцію-делегат, яку коннектор використовує для відправлення подій оновлення з висхідної системи. Використання функції-делегата замість каналу напряму дозволяє провести додаткову обробку повідомлення перед тим, як надіслати його до висхідного коннектора. Наприклад, менеджер перед надсиланням повідомлення додає до нього ідентифікатор висхідної системи та вказує час, коли було сформовано дану подію. Для запуска коннектора передбачено два стартових режими роботи: звичайний старт (StartForStreamOnly) та старт з синхронізацією (StartWithResync). Даний метод блокує горутину на час своєї роботи, тож бажаним є використання окремої горутини для його роботи;

- OnStart – даний метод повинен містити функціонал ініціалізації коннектора перед початком його роботи, наприклад підключенням до висхідної системи; якщо коннектор не може розпочати свою роботу, даний метод поверне помилку;

- OnStop – описує процедуру звільнення ресурсів, що були створені в методі OnStart; при виникненні проблем даний метод поверне помилку.

```
type Connector interface {
    // OnStart performs preparations which must be done before streaming started
    OnStart() error

    // OnStop cleanups resources created in OnStart
    OnStop() error

    // StartStreaming generate events to create, update or deleting objects
    // when corresponding changes occur in the source database
    // When mode is StartWithResync generates update events for all db entities
    // StartWithForceResync is similar to StartWithResync but also generates cleanup event before updates
    StartStreaming(events <-chan ConnectorEvent, mode StartMode, target SubmitFunc) error
}
```

Рисунок 7.1 – Інтерфейс коннектора

Даний інтерфейс передбачає в собі лише основний функціонал для роботи коннектора, та не містить методів для його побудови та оновлення стану. Для розв'язку даної проблеми використовується інтерфейс конструктора (Builder). Його основним призначенням є формування коннектора та оновлення його конфігурації. Реалізації даного інтерфейсу повинні включати в себе конфігурацію, необхідну для з'єднання з висхідною системою, та описувати як передати в коннектор конфігурацію висхідної системи, та яким чином передати коннектору налаштування елементів мережі.

Інтерфейс конструктора представлено на рис.7.2. Він складається з наступних елементів:

- Build – описує функціонал створення коннектора з конфігурації висхідної системи та конфігурації елементів мережі. Якщо коннектор не може бути створеним, даний метод поверне помилку;
- UpdateConfig – оновлює конфігурацію коннектора. Якщо даний конструктор не є сумісним з переданим коннектором, або виникне помилка при оновленні коннектора, даний метод поверне помилку. Варто зазначити, що даний метод не передбачає зупинку роботу коннектора на час оновлення конфігурації, натомість синхронізація роботи коннектора повинна бути виконана зовнішніми механізмами. Якщо конфігурацію не може бути оновлено, даний метод поверне помилку.

```
type Builder interface {
    Build(config []model.ConfigPart) (Connector, error)
    UpdateConfig(config []model.ConfigPart, c Connector) error
}
```

Рисунок 7.2 – інтерфейс конструктора

Менеджер є останнім елементом бібліотеки. Його завданням є автоматизація життєвого циклу коннектора, координація оновлення конфігурації та обробки подій синхронізації, надання зручного інтерфейсу для роботи з коннектором. Менеджер повністю абстрагує від розробника роботу з коннектором, натомість уся взаємодія

здійснюється через інтерфейси-конструктори. Бібліотека містить також готову реалізацію менеджера, яка є оптимальною для використання у більшості випадків. Також дана реалізація включає в себе відмовостійкість – якщо при старті чи оновленні роботи коннектора виникне помилка, система буде перезапускати коннектор з перервами в певний інтервал допоки коннектор не почне свою роботу. Проте є бажаною робота з менеджером через інтерфейс, що дасть змогу легко проводити блочне тестування компонентів, що його використовують.

Інтерфейс менеджера представлено на рис.7.3. Він складається з наступних методів:

- Restart – даний метод відповідає за перезапуск коннектора, що означає повну зупинку існуючого коннектора, звільнення всіх використовуваних ресурсів та повне проходження процедури створення нового коннектора, який замінить існуючий; даний метод поверне помилку за виникнення проблем при перезапуску коннектора;

- FullSync – описує функціонал синхронізації коннектора. На відміну від перезапуску, даний метод зберігає існуючий коннектор. Натомість, він зупиняє коннектор, оновлює його конфігурацію та поновлює його роботу. При виникненні проблем синхронізації коннектора, даний метод поверне помилку;

- Stop – даний метод зупиняє роботу коннектора та звільняє зайняті ним ресурси. Даний метод повертає помилку якщо виникне проблема при звільненні зайнятих ресурсів.

```
type Manager interface {
    Restart() error
    FullSync(force bool) error
    Stop() error
}
```

Рисунок 7.3 – інтерфейс менеджера

В бібліотеку streaming-core також входить модель подій оновлень, які надсилають коннектори. Дана модель представляє собою основний контракт для

даних, якими обмінюються коннектори та nvt-composer. Модель включає в себе наступні елементи: тип події, вид об’єкту, що спричинив подію, та структура даного об’єкту. Опис структури об’єкту наведено в таблиці 7.1.

Таблиця 7.1 – Опис структури об’єкту події

| Поле | Тип | Обов’язкове | Опис |
|---|---|---|---|
| id | string | так | унікальний ідентифікатор об’єкта |
| type | string | так | тип об’єкта |
| class | string | - | клас об’єкта |
| name | string | так | ім’я об’єкта |
| meta | AttributeMap | - | мета-інформація про об’єкт (використовується лише при обробці даних) |
| attributes | AttributeMap | - | атрибути об’єкту |
| createdWhen | string | - | час створення об’єкту в висхідній системі |
| updatedWhen | string | - | час оновлення об’єкту в висхідній системі |
| processedWhen | time.Time | так | час обробки даної події сервісом-стрімером |
| sourceSystem | string | так | ідентифікатор висхідної системи, з якої прийшла дана подія |
| links | []EventLink | - | висхідні посилання на інші об’єкти |

За контрактом між сервісами-стрімерами та nvt-composer, структура об’єкта повинна включати в себе висхідні посилання на інші об’єкти. Структура посилань наведена в табл. 7.2.

Таблиця 7.2 – Опис структури посилань на інші об’єкти

| Поле | Тип | Обов’язкове | Опис |
|---|---|---|---|
| id | string | так | ідентифікатор посилання |
| name | string | так | ім’я посилання |
| type | string | так | тип посилання |
| to | string | так | вузол, на який вказує посилання |
| attributes | AttributeMap | - | атрибути |

Окрім абстракцій для організації процесів отримання даних, бібліотека streaming-core містить пакет, що надає засоби для організації процесу передачі даних до системи обміну повідомленнями.

Основною данного пакету є інтерфейс Session, який представляє собою абстрацію висхідного коннектора, метою якого є передача даних до необхідного призначення. Інтерфейс представлено на рис.7.4.

```
type Session interface {
    // Publish incoming events to a MQ server
    Stream(<-chan struct{}, <-chan model.ChangeEvent) error
}
```

Рисунок 7.4 – інтерфейс Session

Даний інтерфейс складається з єдиного методу Stream, що приймає канал для очікування стоп-сигналу, та канал, з якого необхідно зчитувати події та передавати їх до необхідного призначення в залежності від реалізації. Якщо під час роботи висхідного коннектора виникне невідновлювальна помилка, коннектор припинить свою роботу та поверне помилку.

Бібліотека також надає наступні реалізації даного інтерфейсу:

- rabbitSession –висхідний коннектор, який надсилає події до системи обміну повідомлень RabbitMQ;

- logOnlySession – реалізація, що використовується тестуванні роботи коннектора. На відміну від rabbitSession, просто пропускає повідомлення; натомість виводить не лише тип події та ідентифікатор, а повністю згенеровану інформацію про подію, що буде надіслана.

Streaming-core містить основні будівельні блоки для побудови сервісів-стрімерів, проте існує також певний ряд проблем, рішення яких потребують також інші сервіси системи, такі як nvt-composer та nvt-be. Метою бібліотеки nvt-framework і є виділення в окремий модуль рішень проблем, що є затребуваними всіма сервісами в системі.

Однією з таким проблем є робота з конфігурацією системи. Дана конфігурація використовується усіма сервісами в системі для певних потреб:

- сервіси-стрімери можуть визначити, чи існує конфігурація даного елементу мережевої інфраструктури в системі, і якщо ні, то не надсилати повідомлення про оновлення даного елементу в висхідній системі, зменшуючи навантаження на систему обміну повідомленнями;

- nvt-composer використовує конфігурацію при формуванні графу, а саме інформацію про елемент мережі, як він повинен відображатися на графі, як визначити його групу, чи даний елемент повинен бути заміненим на інший на графі, як він може бути пов’язаним з іншими елементами тощо;

- nvt-be підтримує можливість надання заміни стандартного запиту для отримання графу на інший.

Для забезпечення даної потреби spog-framework містить підмодуль, що представляє механізми отримання конфігурації. Робота з джерелами конфігурації здійснюється через інтерфейс Loader, який описує контракт отримання конфігурації.

Інтерфейс Loader зображено на рис.7.5 та складається з наступних методів:

- Load – описує функціонал отримання конфігурації. Даний метод приймає єдиним аргументом контекст, який може містити дані в рамках запиту, що може бути використаним певними реалізаціями інтерфейсу. Якщо виникне проблема при завантаженні конфігурації, даний метод поверне помилку.

```
type Loader interface {
    Load(ctx context.Context) ([]Part, error)
}
```

Рисунок 7.5 – інтерфейс Loader

Бібліотека містить наступні реалізації даного інтерфейсу:

- fileConfig – дозволяє працювати з локальною конфігурацію, що знаходиться в файлі. Дана реалізація зазвичай використовується на етапі розробки, коли сервіс конфігурації або не є необхідним для перевірки функціоналу, або взагалі не є доступним в необхідний момент;

- serverConfig – реалізує роботу з сервером конфігурації. В якості необов'язкового аргументу може також приймати реалізацію транспорту, що дозволяє користувачу даної реалізації додати можливість передавання додаткових заголовків, налаштувати трасування запитів тощо.

При реалізації провідника конфігурації з сервісу конфігурації був використаний один з широко використовуваних паттернів мови Go «functional options»[13], який дозволяє надати зручний для розробника інтерфейс роботи з необов'язковими параметрами. Даний підхід дозволяє представити необов'язкові параметри у вигляді функцій-опцій, які змінюють структуру конфігурації за замовчуванням. Використання такого підходу дозволяє отримати наступний ряд переваг:

- можливість визначення оптимальних значень за замовчуванням для необов'язкових параметрів;

- гнучкість налаштування (користувач використовує лише ті опції, які йому необхідно налаштувати);

- можливість масштабування з часом (додавання нових опцій не зламає вже написаний код, що використовує даний метод);

- самодокументування параметрів;

- безпечний для розробників, що вперше використовують даний підхід (вони можуть опиратися на налаштування за замовчуванням);

- не потребує використання nil-значень або порожніх значень для примітивних типів для того щоб дотриматись вимог компілятора.

Фрагмент реалізації даного шаблону наведено на рисунках 7.6-7.7. На даному прикладі зображена структура функцій-опцій, що складається з єдиної опції типу http.RoundTripper, яка дозволяє користувачу підмінити стандартний транспорт власним. При цьому, якщо в майбутньому буде додано новий необов’язковий параметр, існуючим користувачам бібліотеки не доведеться змінювати вже існуючий код взаємодії з провідником конфігурації.

```
type Options struct {
    rt http.RoundTripper
}

type Option func(*Options)

func RoundTripper(rt http.RoundTripper) Option {
    return func(opts *Options) {
        opts.rt = rt
    }
}

type serverConfig struct {
    configUrl url.URL
    client    *http.Client
}
```

Рисунок 7.6 – структура провідника конфігурації

## 7.2 Отримання подій про оновлення даних з бази PostgreSQL

Для отримання подій з БД PostgreSQL використовується механізм LISTEN/NOTIFY. На відміну від класичних рішень з використанням WAL, даний підхід є менш виснажливим в плані навантаження ресурсів системи та представляє більш зручний для розробників інтерфейс для отримання інформації про зміни даних в БД.

NOTIFY забезпечує простий механізм міжпроцесорного зв'язку для набору процесів, що використовують спільну БД. Інформація, що передається клієнтові для сповіщення, включає ім'я каналу сповіщень, PID процесу сесії, що викликав сповіщення та опціональний рядок корисного навантаження.

```
func New(configUrl url.URL, setters ...Option) config.Loader {
    opts := &Options{
        rt: http.DefaultTransport,
    }
    for _, setter := range setters {
        setter(opts)
    }
    return &serverConfig{
        configUrl: configUrl,
        client: &http.Client{
            Transport: opts.rt,
        },
    }
}
```

Рисунок 7.7 – використання функцій-опцій при побудові провідника конфігурації

Для узгодження різного типу сповіщень, проектувальник БД повинен визначити назву каналу, використовуючи яку програми зможуть підписатися на необхідні їм події. Існує безліч підходів до найменування каналів. Якщо канал відповідає за оновлення однієї таблиці, він може бути названим відповідно таблиці, за змінами якої він стежить. Проте при розробці розглянутої в дисертації системи було вирішено використовувати один канал для відстеження змін у всіх необхідних таблицях, оскільки це дозволить зменшити навантаження на БД, тому що для кожного каналу, який необхідно прослуховувати, потрібно виділяти одне підключення. Інформація про тип події та таблицю, в якій відбулись зміни, передається у рядку корисного навантаження.

Найбільш розповсюдженим способом використання даного механізму є встановлення викликів NOTIFY або вбудованої процедури pg_notify в тригер, який

викликається під час оновлень таблиці[14]. Таким чином, сповіщення відбуваються автоматично коли таблицю змінюється, і розробники не зможуть випадково пропустити виклик NOTIFY.

Однією з переваг яку надає даний механізм є дедуплікація повідомлень. Якщо в один і той самий канал приходять сповіщення з однаковим рядком корисного навантаження в рамках однієї транзакції, сервер бази даних може вирішити доставити лише одне повідомлення. Аналогічно, сповіщення с однаковими рядками корисного навантаження, які прийшли з різних транзакцій, завжди будуть доставлятися як окремі сповіщення.

Механізм NOTIFY надає певний ряд гарантій які роблять його оптимальним для отримання інформації про зміни в БД. За винятком видалення повторюваних сповіщень згідно вказаних умов, NOTIFY гарантує, що сповіщення від однієї конкретної транзакції будуть доставлені в тому порядку, в якому вони були надіслані. Також гарантується, що повідомлення з різних транзакцій будуть доставлені в тому порядку, в якому були здійснені транзакції.

При розробці систем, що використовують механізм NOTIFY/LISTEN одним з головним аспектів, якому треба приділити увагу, є мінімізація областей, де використовується LISTEN. Для збереження необроблених сповіщень в PostgreSQL виділяється черга повідомлень (8GB за замовчуванням). Якщо одна з сесій виконує команду LISTEN та потім починає транзакцію на дуже довгий час, БД не зможе виконати очистку черги повідомлень допоки дана сесія не завершить транзакцію. Також важливим аспектом є обсяг інформації, що буде передано в рядку корисного навантаження, оскільки рядок корисного навантаження має обмеження на 8000 байт.

Для отримання даних через механізм LISTEN/NOTIFY було розроблено сервіс-стрімер pg-graph-streamer з реалізацією коннектора, що використовує одне з’єднання для прослуховування повідомлень на заздалегідь узгодженому каналі сповіщень. Прослуховування сповіщень відбувається в окремій горутині, де вони зчитуються та через механізм каналів мови Go передаються основній горутині, яка оброблює їх та передає у висхідний коннектор, який у свою чергу надсилає їх у RabbitMQ.

## 7.3 Отримання подій про оновлення бази даних з MongoDB

MongoDB підтримує можливість спостереження за змінами в базі даних через oplog. Oplog представляє собою спеціальний тип коллекції (capped collection), яка зберігає з ротацією записи всіх операцій що змінюють дані в БД.

Під час виконання операцій з базою даних, MongoDB спершу застосовує операції на голові реплікаційного набору, а потім записує операції в oplog голови. Потім інші члени реплікаційного набору копіюють та застосовують ці операції в асинхронному режимі. Усі члени реплікаційного набору мають копію oplog-а в колекції local.oplog.rs, яка дозволяє їм підтримувати поточний стан бази даних.

Кожна операція в oplog є ідемпотентною. Тобто операції oplog дають однакові результати, при застосовувані один раз або декілька разів до цільового набору даних.

Отримання подій з MongoDB реалізовано в сервісі nvt-service-streamer, який містить коннектор, що використовує oplog. Для роботи з oplog було використано бібліотеку gtm. Дана бібліотека надає зручний API для роботи з oplog, та підтримує можливість зчитати всі документи напряму при старті для обробки підходом, аналогічним роботі з oplog. Коннектор отримує інформацію про оновлення в БД через канал сповіщень, що надає gtm.

## 7.4 Реалізація процесу побудови графа

Структура графа мережевої інфраструктури передбачає розділення його елементів на різні типи в залежності від їхньої ролі. Множина типів включає в себе наступний набір значень:

- VISIBLE – звичайний тип елемента; вказує на те, що даний тип елемента повинен бути відображеним на графі;

- TRANSPARENT – елемент-посередник; не повинен відображатися на графі, проте замість нього на основі механізму посилань заміни буде обраховано елемент, який повинен бути присутнім на граф;

- INTERNAL – елемент напряму не присутній на графі, проте може використовуватися при отриманні графу через сервіс nvt-be;

- GROUP – група, даний елемент є контейнером, який може містити у собі набір інших елементів.

Кожен з типів елементів містить власну логіки, за якої вузол з даним типом повинен бути обробленим, проте також для всіх типів існує частина спільної логіки, яка використовується при обробці вузла. Для оптимального рішення даної проблеми логіка обробки вузла була побудована з використанням шаблону програмування Декоратор.

Декоратор – структурний шаблон програмування, призначення якого є динамічне підключення додаткової поведінки до існуючого об’єкту[15]. На відміну від традиційних рішень з використанням наслідування класів (у випадку мови Go – композицією структур), даний шаблон дозволяє досягти гнучкості, оскільки нова функціональність не потребує наслідування класу, що в свою чергу не призведе до обмеження об’єкту, що реалізовує даний функціонал, ієрархією базового класу; також буде усунуто проблему діаманта, оскільки в мові Go при використанні композиції декількох структур, що реалізовують метод з однаковою сигнатурою, компілятор не зможе визначити, який з методів необхідно буде викликати.

Схема реалізації шаблону Декоратор представлена на рисунку 7.8. Основною реалізації шаблону декоратор є інтерфейс Handler, який містить метод Handle, який описує процедуру обробки вузла та метод CanEnter, який описує чи може даний обробник бути використаним для обробки події. В свою чергу, він містить ряд реалізацій, які компонуються між собою для досягнення бажаної послідовності обробки вузла:

- rootHandler – головний обробник, містить у собі конфігурацію усіх елементів графу, на основі якої перенаправляє подію до необхідного обробника в залежності від її типу; якщо для події не існує конфігурації, її буде проігноровано;
- visibleHandler – обробник, що оброблює вузли з типом VISIBLE;
- transparentHandler – обробник, що оброблює вузли з типом TRANSPARENT;
- groupHandler – обробник, що оброблює вузли з типом GROUP.

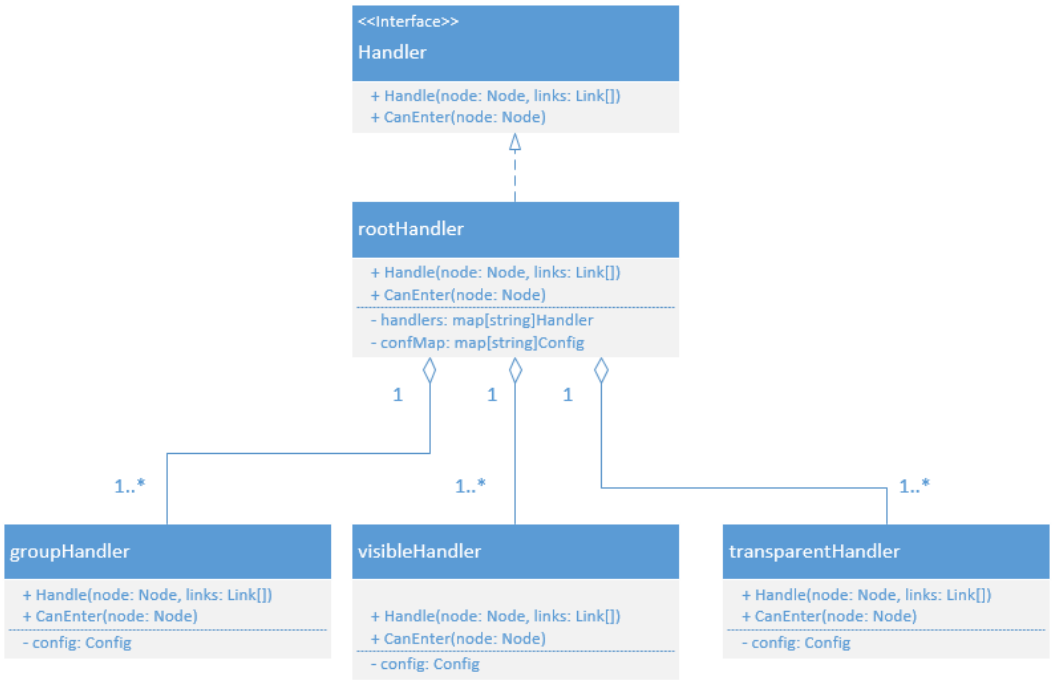


Рисунок 7.8 – діаграма класів для реалізації шаблону Декоратор

## 7.5 Розробка мови опису шляхів до вузла в графі

Можливість ефективного проходу по графу є однією з найбільш поширених задач при роботі з графовими БД. Для виконання даної задачі в ArangoDB використовується мова запитів AQL, яка дозволяє однією мовою запитів працювати з усіма підтримуваними моделями БД. Дана мова володіє широкими можливостями для пошуку графів, проте її синтаксис є занадто громіздким для написання запитів пошуку в графі. Це не є проблемою якщо всі запити зосереджені в програмному коді системи, проте якщо система підтримує можливість використання користувацьких запитів, даний недолік стає більш очевидним.

Саме тому було вирішено розробити предметно-орієнтовану мову програмування для вирішення задачі обходу графа. Основні вимоги, які було висунуто до мови:

- лаконічність;
- простота;
- підтримка пошуку по прямим та оберненим посиланням.

Мова GQL (Graph Query Language) представляє собою DSL, побудований на основі підмножини відкритої специфікації openCypher та пропонує наступні особливості:

- простий та ергономічний синтаксис;
- пошук вузлів по типам;
- пошук вузлів по значенням атрибутів;
- пошук по прямим та оберненим посиланням.

Порівнянно з мовою AQL, мова GQL дозволяє описувати обхід графу більш компактним та зрозумілим синтаксисом. Для порівняння було обрано один простий запит написаний на AQL та GQL. Порівняння наведено на рисунку 7.9.

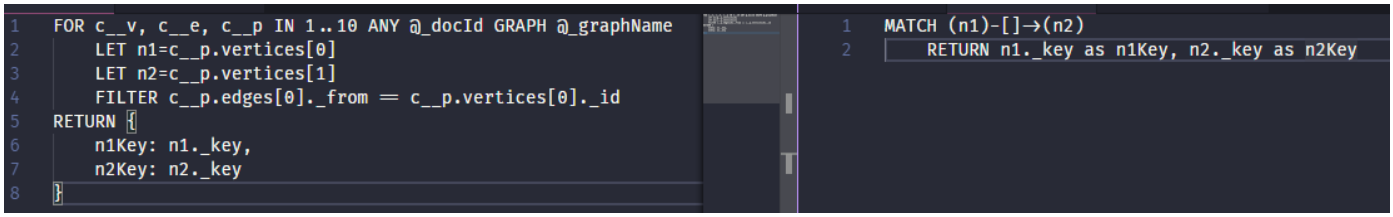


Рисунок 7.9 – порівняння запиту на обхід графів
(зліва – запит написаний мовою AQL, справа – GQL)

Парсер для мови GQL було розроблено з використанням підходу написання парсерів під назвою parser combinators. Парсери-комбінатори – широко відомий підхід для створення парсерів, в якому на відміну від традиційного підходу генерації коду на основі описання граматики мови (yacc, bison, ANTLR) використовуються функції вищого порядка для побудови невеликих парсерів, з композиції яких і утворюється результуючий парсер мови[16]. Перевагою такого методу є використання однієї мови для опису граматики мову, та власне самого парсера.

Окрім парсера також розроблено генератор запитів, що перетворює правильно сформований запит мови GQL у запит мови AQL.

## 7.6 Алгоритм формування графу

Згідно п.7.4, кожна подія про зміну елемента в системі проходить обробку, перед тим як потрапити до графу, проходить через визначений для типу вузла обробник. Проте для всіх обробників існує спільна частина, яка окрім створення вузла та висхідних посилань, включає в себе створення посилань на групу, посилання для елементи-замінники та посилання на основі обчислюваних атрибутів. В основі даних процедур лежить схожий алгоритм поширення даних по ієрархії елементів. Загальну схему алгоритму можна описати наступним чином:

- пошук конфігурації посилання, за яким здійснювати поширення (для групових посилань – секція GroupLink в моделі конфігурації, для посилань заміни – ReplaceLink);
- перевірка, чи є в графі дане посилання;
- якщо посилання є, перевірити чи є з’єднаний вузол за даним посиланням;
- якщо з’єднаний вузол є в графі, взяти з нього значення, яке необхідне поширити;
- запустити обробку всіх вузлів, з’єднаних з вузлом, що оброблюється, через вхідні посилання.

Для демонстрації прикладу роботи даного алгоритму представимо модель частини графа, що складається з VNF (тип елементу - GROUP), двох її компонентів (тип елементу – VISIBLE), кожен з яких має один внутрішній порт (тип елементу - TRANSPARENT), при цьому порт з першої VNF-компоненти зв’язаний з портом другої VNF-компоненти. Оскільки порти мають тип TRANSPARENT, на результуючому графі вони не мають бути показані, а посилання повинні бути перенесені на рівень вище, тобто з однієї VNF-компоненти на іншу. Діаграму синхронізації для описаного сценарію наведено у додатку И.

В процесі отримання даних з систем не існує гарантії, що події прийдуть в тому порядку, в якому порядку прийдуть елементи мережевої інфраструктури, і це необхідно враховувати при поширенні значень. В даному прикладі буде розглянута робота алгоритму за наступного порядку отримання подій:

- отримання VNF;
- отримання PORT1;
- отримання VNFC1;
- отримання VNFC2;
- отримання PORT2.

За даного набору подій алгоритм обробить їх наступним чином:

а) отримання VNF:

1) конфігурація не містить посилань для поширення значень;
2) додаємо вузол в граф.

б) отримання PORT1:

1) знаходимо конфігурацію, за якою необхідно знайти посилання, через яке знайти елемент для заміни даного елемента (VNFC1);
2) посилання існує, тож виконуємо пошук зв'язаного вузла за посиланням;
3) з’єднаного вузла (VNFC1) не існує на даний момент в графі, тож на даний момент неможливо поширити значення;
4) додаємо вузол в граф.

в) отримання VNFC1:

1) знаходимо конфігурацію, за якою необхідно знайти посилання, через яке знайти елемент, за яким буде визначено групу, до якої належить даний елемент (VNF);
2) посилання існує, тож шукаємо зв’язаний елемент за посиланням;
3) з’єднаний вузол існує у графі, отже вказуємо для вузла групу VNF;
4) додаємо вузол в граф;
5) запускаємо обробку для вузлів, з’єднаних вхідним посиланням (PORT1);
6) для PORT1 виконуємо кроки 1-2 пункту б;
7) з’єднаний за посиланням з PORT1 вузол (VNFC1) існує, тож встановити для вузла елементом-замінником VNFC1.

г) отримання VNFC2:

1) знаходимо конфігурацію, за якою необхідно знайти посилання, за яким буде визначено групу, до якої належить елемент (VNF);

2) посилання існує, тож шукаємо зв’язаний елемент за посиланням;

3) з’єднаний вузол (VNF) існує у графі, отже вказуємо для групу VNF;

4) додаємо вузол в граф.

д) отримання PORT2:

1) знаходимо конфігурацію, за якою необхідно знайти посилання, через яке знайти елемент, який буде використано для заміни даного елемента (VNFC1);

2) посилання існує, тож виконуємо пошук зв'язаного вузла за посиланням;

3) з’єднаний за посиланням з PORT2 вузол (VNFC2) існує, тож встановити для вузла елементом-замінником VNFC2;

4) додаємо вузол в граф.

Виходячи з описаних вище кроків роботи алгоритму можна побачити, що незважаючи на неправильний порядок, за яким прийшли події, значення було поширено правильно, а тому посилання будуть перенесені згідно описаної конфігурації. Схематичне представлення даного прикладу зображено в Додатку 6.

## 7.7 Синхронізація графу з актуальним станом мережевої інфраструктури

Система nvt реалізовано таким чином, що всі оновлення елементів в висхідних системах, що вказані в конфігурації, буде передано в nvt-composer та після обробки додано в граф, тобто система nvt надає гарантію тимчасової консистентності Проте існують такі ситуації, внаслідок яких тимчасово система не може проводити спостереження за станом висхідної системи (коннектор було запущено вже після старту висхідної системи, коннектор було тимчасово вимкнено внаслідок технічних причин тощо). Такі ситуації призводять до порушення консистентності, що може в свою чергу призвести до хибного відображення даних на клієнтській частині.

Засобом вирішення даної проблеми є спеціальний тип події – синхронізація (RESYNC). Дану подію коннектор може надіслати в чергу повідомлень, вказавши в полі CreatedWhen час в який було створено дану подію. Стрімери перед посиланням

даного сигналу посилають «події-заглушки» про оновлення усіх елементів з висхідної системи.

При отриманні даної події сервіс nvt-composer виконає запит на видалення всіх елементів, які були оброблені стрімером раніше ніж була створена дана подія та які прийшли з висхідної системи, за яку відповідає стрімер. Саме тому поля ProcessedWhen та SourceSystem в моделі подій є обов'язковими, оскільки вони дозволяють встановити застарілі вузли які залишилися внаслідок порушення консистентності між nvt та висхідною системою.

Подія синхронізації завжди запускається при старті коннектора з метою відновлення консистентності. При проектуванні коннектора бажаним є виконання обробки повідомлень та синхронізації в одній горутині з метою уникнення стану гонок подій (data races). Наприклад, якщо обробка повідомлень та синхронізація виконуються паралельно, та подія надсилається під час синхронізації, то вона може бути перезаписаною іншою подією того ж самого елемента, яка була сформована під час синхронізації.

## 7.8 Механізм сповіщень про оновлення графу

Взаємодія між клієнтською та серверною частиною забезпечується через REST API сервісу nvt-be. Проте саме лише використання REST API не є оптимальним для реалізації сповіщень про оновлення графу. Для вирішення даної проблеми використовується протокол, що забезпечує обмін повідомлень між браузером та веб-сервером в режимі реального часу.

Повідомлення в системі відбувається наступним чином:

- при встановленні WebSocket каналу між клієнтом та сервером, сервер надсилає стартове повідомлення, яке містить граф мережевої інфраструктури залежно від обраної клієнтом стартової точки;

- при оновленні графу сервер надсилає клієнту повідомлення про оновлення графу, яке містить перелік доданих та оновлених елементів графу - вузлів, груп та посилань; та набір ідентифікаторів для видалених елементів графу.

Для спостереження за графом на серверній частині використовується механізм поллінгу за визначеним в конфігурації запитом, проте при подальшому розвитку системи даний механізм може бути замінений на використання WAL-логу для отримання сповіщень про зміни в базі.

## 7.9 Реалізація моделі конфігурації елементів графу

В якості моделі для конфігурації елементів графу використовується модель даних EAV. Дана модель є аналогом ефективних в плані використання ресурсів системи методів зберігання розріджених значень, де зберігаються лише непорожні значення. У моделі EAV кожна пара атрибутів-значень – це факт, що описує сутність, а рядок у таблиці EAV зберігає один факт[17].

В мінімальному представленні дані у моделі EAV записуються у три стовпці:

- сутність – предмет, що описується (в випадку nvt-config – тип елемента);
- атрибут – зазвичай реалізується у вигляді зовнішнього ключа у таблиці визначень атрибутів.

Адаптери для обчислення значень застосовуються в процесі обробки подій сервісом nvt-composer. Процес формування графа включає в себе формування вузлів та ребер, що їх зв’язують. Вузли графа складаються з атрибутів, які приходять в сповіщеннях про створення або оновлення елементу графа. Проте існують також такі атрибути, які потребують виконання додаткових розрахунків. Аналогічна ситуація виникає зі створенням зв’язку між вузлами. Деякі зв’язки можна отримати з системи, проте існують такі зв’язки, в яких вузол на який треба зробити посилання, необхідно обраховувати. Сервіс nvt-composer використовує конфігурацію обчислюваних атрибутів для того щоб утворити обчислювальні функції, які будуть виконуватися при обробці події про оновлення вузла графа.

## 7.10 Висновки до розділу

В даному розділі були розглянуті основні питання, які виникли на етапі реалізації системи. Хоча мікросервісна архітектура передбачає розділення системи на окремі мікросервіси, виникає ряд проблем, що потребують рішення для більшості сервісів в системі, такі як робота з конфігурацією або побудова сервісу-стрімера. Рішення таких питань було винесено в окремі модулі: streaming-core, який описує основні контракти, яких повинен дотримуватись сервіс-стрімер, та nvt-framework, який представляє механізм для конфігурації системи.

Головним завданням коннектора є отримання даних з висхідних систем та передача повідомлень про зміни їх стану. Для виконання даної задачі необхідно передбачити механізм ефективного збору інформації. Було розглянуті способи отримання повідомлень про зміни стану для баз даних PostgreSQL та MongoDB.

Пояснено основні алгоритми, що використовуються сервісом nvt-composer для побудови графу мережевої інфраструктури. Продемонстровано на прикладі простої моделі графу механізм поширення значення в ієрархії.

Наведено основні переваги використання моделі даних EAV для збереження конфігурації.

# 8 РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

При розробці користувацького інтерфейсу користувача висувається ряд вимог, яким має відповідати клієнт: зручність графічного інтерфейсу, консистентний вигляд, використання оптимальної кольорової гами тощо. В даному розділі наведено основні питання, які виникли під час розробки, та опис їх рішення.

## 8.1 Використання Material Design

Для розробки загального дизайну системи було обрано підхід до проектування дизайнеру графічних інтерфейсів Material Design, який характеризується широким використанням суворих макетів, анімацій та переходів, відступів та ефекту глибини світла та тіні. За ідеєю дизайнерів Google, застосунок не повинен мати гострих рамок, а переходи між сторінками інтерфейсу повинні відбуватися між собою плавно та практично непомітно[18].

Реалізація стилю Material Design потребує дотримання набору вимог, описаних Google на офіційному сайті в розділі «Guidelines». Саме тому було використано фреймворк Vuetify. який представляє собою реалізацію стилю Material Design та надає користувачу набір компонентів, на основі яких буде побудовано користувацький інтерфейс.

## 8.2 Використання модуля для управління станом Vuex

Клієнтська частина системи використовує мову програмування TypeScript та фреймворк VueJS. VueJS дозволяє описати стан інтерфейсу системи через декларативну модель компонентів. Використання базової частини бібліотеки є оптимальним для розробки додатків малої та середньої складності, проте при розробці великих систем можуть виникнути проблеми масштабування кодової бази.

Модуль Vuex представляє собою бібліотеку для керування станом клієнтської частини та набір практик для організації даних. Додатки, що використовують бібліотеку Vuex, зазвичай впроваджують архітектуру MVVM для відділення шару роботи з даними від логіки побудови графічного інтерфейсу. Vuex при цьому виконує роль моделі, яка містить у собі всю логіку взаємодії з серверною частиною, а Vue бере на себе роль ViewModel, виступаючи посередником між моделлю та представленням. Схема архітектури MVVM з використанням бібліотеки Vuex зображено на рисунку 8.1.

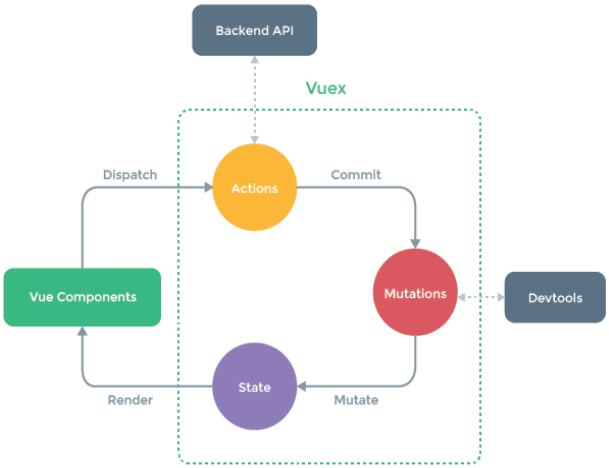


Рисунок 8.1 – типова архітектура додатку на основі Vuex

## 8.3 Висновки до розділу

Дизайн системи є однією з ключових проблем при проектуванні клієнтської частини. Використання стилю Material Design дозволяє розробити дизайн системи на основі чіткого набору критеріїв, яким повинен повідати зовнішній вигляд системи.

Бібліотека Vuex дозволяє запровадити чітку архітектуру клієнта, що забезпечує легке масштабування клієнтських додатків високого рівня складності.

# 9 ТЕСТУВАННЯ СИСТЕМИ

Головною метою етапу тестування системи є забезпечення інформації про якість програмного продукту, об’єктивну та незалежно оцінку правильності реалізації програмного забезпечення, дозволити учасникам процесу розробки системи визначити та зрозуміти основні дефекти реалізації програмного продукту.

## 9.1 Модульне тестування системи

Модульне тестування системи є одним з методів тестування системи, в якому один або більше компонентів програмної системи разом з пов’язаними з ними даними проходять перевірку на придатність до використання. Модульні тести зазвичай представляють собою автоматизовані тести, написані розробниками системи для перевірки секції застосунку на відповідність очікуваній поведінці в можливих ситуаціях використання.

Тестування програмного забезпечення передбачає динамічну перевірку програми на скінченній множині тестових даних, відібраних певним чином з нескінченного вхідного простору, для перевірки на відповідність очікуваній поведінці системи[19].

Саме тому на етапі тестування системи виникає проблема вибору обмеженої множини тестів, оскільки написання вичерпних тестів для перевірки всієї множини вхідних даних або є неефективним з точку зору використаного часу або взагалі не є можливим за наявності побічних ефектів в модулі, який необхідно протестувати. Для системи було обрано ряд критичних програмних компонентів, що містять у собі нетривіальну логіку роботи, та проведено їх покриття модульними тестами.

Написання модульних тестів було здійснено з використанням бібліотеки testify, що працює на основі засобів написання та запуску тестів зі стандартної бібліотеки. Testify розширює стандартний функціонал набором функцій-утиліт, що

дозволяють підвищити читабельність написаних тестів та позбавитись певних повторюваних фрагментів коду.

Компоненти, що складають програмний продукт, не працюють ізольовано один від одного, а взаємодіють один з одним для забезпечення роботи системи, проте модульне тестування передбачає перевірку окремих частин системи незалежно від влаштування компонентів, що взаємодіють з ними. Для ізоляції залежностей при написанні модульних тестів бібліотека testify надає функціонал для побудови об’єктів-заглушок, проте його використання передбачає написання великої кількості повторюваних фрагментів коду. Рішенням даної проблеми може бути генерація коду об’єктів-заглушок для інтерфейсу або використання рефлексії. В мові Go надають найкращою практикою є використання засобів генерації коду, саме тому було використано утиліту mockery, яка дозволяє легко згенерувати код об’єкту-заглушки для вказаних інтерфейсів, що в свою чергу позбавляє розробника необхідності написання великої кількості інфраструктурного коду.

## 9.2 Висновки до розділу

Тестування системи є невід’ємною складовою процесу програмної інженерії. Ключовим аспектом в процесі тестування є вибір скінченного набору даних, який повинні охопити основні сценарії використання модуля системи, та написання для нього тестів для перевірки відповідності очікуваній поведінці.

Використання модульного тестування дозволило визначити правильність роботи критичних частин програмного продукту. При написанні модульних тестів було використано допоміжні бібліотеки для забезпечення ізоляції компоненту, який перевірявся, від його залежностей.

# 10 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Стартап представляє собою реалізацію новаторської ідеї, орієнтовану на комерційний успіх. На новаторстві ідеї робиться головний акцент, розроблюваний продукт повинен привнести новизну в існуючій галузі. Запровадження нової ідеї завжди несе у собі ризик, однак на даний час існує безліч засобів для отримання фінансування стартапів (стартап-майданчики, бізнес-інкубатори тощо). Проте разом з тим для нових стартапів висується ряд вимог, яких необхідно відповідати для того щоб залучити необхідний рівень фінансування. Окрім нової ідеї також повинні враховуватися її подача, підхід до реалізації.

## 10.1 Опис ідеї проекту

Таблиця 10.1 – Опис ідеї стартап-проекту

| Зміст ідеї | Напрями використання | Вигоди для користувача |
|---|---|---|
| | 1. Середовище для спостереження за станом мережі | Спрощення процесів, пов’язаних з аналізом стану мережевої інфраструктури |
| | 2. Візуалізація карти мережі | Наочне зображення елементів мережі, та їх взаємозв’язків |

Основні техніко-економічні характеристики ідеї:

- наявність клієнта на всіх основних платформах;
- автоматична побудова карти мережі;
- відображення оновлень стану в мережі в режимі реального часу;
- широкі можливості збору даних про елементи в мережі;
- можливість регулювання рівня відображення в мережі;
- наявність вбудованих засобів для моніторингу;

- можливість налаштування процесу формування графу;
- засоби інтеграції з іншими системами моніторингу.

Основними конкурентами можна визначити три системи, що виконують аналогічні функції: система моніторингу Zabbix, програмування для спостереження стану мереж LanState та система моніторингу TheDude. Для визначення рівня конкурентоспроможності розроблюваною системи, були визначені її сильні, слабкі та нейтральні характеристики шляхом визначення конкурентів та порівняльного аналізу їх техніко-економічних характеристик. Результат порівняння розроблюваної системи с системами-конкурентами представлено в таблиці 10.2.

Таблиця 10.2 – Сильні, слабкі та нейтральні характеристики проекту

| № | Техніко-економічні х-ки ідеї | Потенційні товари/концепції конкурентів | | | | W (слабка с-на) | N ( нейтральна с-на) | S (сильна с-на ) |
|---|---|---|---|---|---|---|---|---|
| | | nvt | Zabbix | LanState | The Dude | | | |
| 1. | Наявність клієнта на всіх основних платформах | Є | - | - | - | + | - | - |
| 2. | Автоматична побудова карти мережі | Є | - | Є | Є | + | - | - |

Продовження таблиці 10.2

| № | Техніко-економічні х-ки ідеї | Потенційні товари/концепції конкурентів | | | | W (слабка с-на) | N (нейтральна с-на) | S (сильна с-на) |
|---|---|---|---|---|---|---|---|---|
| | | nvt | Zabbix | LanState | The Dude | | | |
| 3. | Відображення оновлень стану в мережі в режимі реального часу | Є | - | Є | Є | + | - | - |
| 4. | Широкі можливості збору даних про елементи в мережі | Є | Є | - | Є | - | + | - |
| 5. | Можливість регулювання рівня відображення в мережі | Є | Є | - | Є | + | - | - |
| 6. | Наявність вбудованих засобів моніт-гу | - | Є | Є | Є | - | - | + |

Продовження таблиці 10.2

| № | Техніко-економічні х-ки ідеї | Потенційні товари/концепції конкурентів | | | | W (слабка с-на) | N ( нейтральна с-на) | S (сильна с-на ) |
|---|---|---|---|---|---|---|---|---|
| | | nvt | Zabbix | LanState | The Dude | | | |
| 7. | Можливість налаштування процесу формування графу | Є | - | - | - | + | - | - |
| 8. | Засоби інтеграції з іншими системами моніторингу | Є | - | - | Є | - | + | - |

## 10.2 Технологічний аудит ідеї проекту

На етапі проектування проекту необхідно провести аналіз технологій, які будуть використані при його реалізації. В результаті було визначено стек технологій, які будуть використовуватися, їх доступність. Таким чином, була здійснена технологічна здійсненність ідеї проекту. Результати технологічного аудиту представлено в табл.10.3.

В результаті проведення аналізу було визначено, що необхідні для реалізації проекту технології є доступними для використання, знаходяться у вільному доступі; методи реалізації є доступними.

Таблиця 10.3 – аналіз технологічної здійсненності проекту

| № | Ідея проекту | Технології її реалізації | Наявність технології | Доступність технології |
|---|---|---|---|---|
| 1. | Збереження графу мережевої інфраструктури | ArangoDB | Наявна | Доступна |
| 2. | Обмін повідомленнями про оновлення в системі | RabbitMQ | Наявна | Доступна |
| 3. | Збереження конфігурації елементів графу | PostgreSQL | Наявна | Доступна |
| 4. | Обмін повідомленнями між клієнтом та сервером в реальному часі | WebSocket | Наявна | Доступна |
| 5. | Реалізація серверної частини | Golang | Наявна | Доступна |
| 6. | Реалізація клієнтської частини | мова JavaScript; фреймворк VueJS | Наявна | Доступна |
| Обрана технологія реалізації ідеї проекту: є можливою. | | | | |

## 10.3 Аналіз ринкових можливостей стартап-проекту

Перед виходом стартапу на ринок, необхідно визначити основні ринкові загрози, спланувати напрями розвитку, визначити поточний стан ринку, потреби потенційних клієнтів та пропозиції конкурентів. Потенційна характеристика ринку представлена на таблиці 10.4.

Таблиця 10.4 - Попередня характеристика потенційного ринку

| № | Показники ринку (найменування) | Характеристика |
|---|---|---|
| 1. | Кількість головних гравців, од | 3 |
| 2. | Загальний обсяг продаж, грн./ум.од. | Понад 5000 |
| 3. | Динаміка ринку | Зростає |
| 4. | Наявність обмежень для входу | Необхідність надання засобів інтеграції з існуючими системами моніторингу |
| 5. | Специфічні вимоги до стандартизації та сертифікації | - |
| 6. | Середня норма рентабельності в галузі (або по ринку), % | 125 |

Згідно результатів проведеного дослідження, ринок є привабливим для входження, а норма рентабельності є задовільною.

Наступним кроком є визначення потенційних клієнтів, їх характеристики та сформувати перелік вимог до кожного товару. Характеристика представлена в таблиці 10.5.

Таблиця 10.5 – Характеристика потенційних клієнтів стартап-проекту

| Потреба, що формує ринок | Цільова аудиторія (цільові сегменти ринку) | Відмінності у поведінці різних потенційних цільових груп клієнтів | Вимоги споживачів до товару |
|---|---|---|---|
| Потреба в системі відображення актуального стан мережі | Користувачі систем моніторингу | Використання нових технологій при розробці, широкі можливості інтеграції | - висока якість продукту;<br>- автоматична побудова графу мережі; |

Продовження таблиці 10.5.

| Потреба, що формує ринок | Цільова аудиторія (цільові сегменти ринку) | Відмінності у поведінці різних потенційних цільових груп клієнтів | Вимоги споживачів до товару |
|---|---|---|---|
| | | | - інтеграція з існуючими системами;<br>- оновлення графу при відображенні в реальному часі. |

Після проведення аналізу потенційних груп клієнтів, було визначено що більшість клієнтів вже використовує засоби моніторингу мереж. Тож необхідно приділити більшу увагу їх головним вимогам, в першу чергу забезпечення засобів інтеграції з вже існуючими системами моніторингу мережі. Після визначення основних вимог клієнтів, необхідно провести огляд ринкового середовища, розглянути фактори, що можуть сприяти ринковому впровадження проекту, та факторів, що можуть йому перешкоджати (таблиці 10.6-10.7).

Таблиця 10.6 – Фактор загроз стартап-проекту

| № | Фактор | Зміст загрози | Можлива реакція компанії |
|---|---|---|---|
| 1. | Використання клієнтом нерозповсюджених систем моніторингу | Відсутність засобів інтеграції з системами моніторингу, а отже неможливість отримання даних про стан мережі | Побудова нового сервіса-стрімера за домовленністю з клієнтом |

Продовження таблиці 10.6

| | | | |
|---|---|---|---|
| 2. | Поява нових засобів для здійснення моніторингу мережі | Відсутність інтеграції з новими засобами | Адаптація до нових засобів здійснення моніторингу мережі |
| № | Фактор | Зміст загрози | Можлива реакція компанії |
| 3. | Необхідність відображення на графі додаткових елементів | Додаткові елементи можуть мати статичний характер | Створення нового коннектора для роботи з додатковими елементами |
| 4. | Поява нової системи-конкуренту на ринку | Поява конкуренції | Аналіз продукту конкурента, адаптація проекту |

Таблиця 10.7 – Фактори можливостей стартап-проекту

| № | Фактор | Зміст можливості | Можлива реакція компанії |
|---|---|---|---|
| 1. | Зацікавленість у продукті виробників систем моніторингу | Розширення списку підтримуваних систем моніторингу | Кооперація з метою запровадження офіційно підтримуваних засобів інтеграції з системою моніторингу виробника |
| 2. | Запровадження продукту в невеликих мережах без систем моніторингу | Розширення ринку, нові клієнти | Адаптація програмного продукту для налаштування роботи з системами, що не використовують засоби моніторингу |

Для проекту були встановлені основні фактори загроз та можливостей. Переважна більшість загроз пов’язана з питанням інтеграції системи з вже

існуючими рішеннями, що сприяє проект до постійного аналізу середовища та адаптації до нових гравців на ринку. Проте було встановлено, що наявні засоби інтеграції в системі дозволяють легко вирішити загрози, що призводить до невеликих шансів їх виникнення. Можливості, які сприяють впровадженню продукту на ринку, призводять до розширення можливостей продукту, а отже і цільової аудиторії. Позитивні можливості загалом переважають ризики виникнення загроз. Таким чином, проект має високий потенціал розвиту в ринковому середовищі, а отже наступним кроком аналізу ринкових можливостей стартап проекту є аналіз існуючої конкуренції (табл. 10.8).

Таблиця 10.8 – Ступеневий аналіз конкуренції на ринку

| Особливості конкурентного середовища | В чому проявляється дана характеристика | Вплив діяльності підприємства |
|---|---|---|
| Тип конкуренції: олігополія | Функціонал відображення графу мережі прив’язаний до системи моніторингу | Створення окремого рішення для покращення загальної якості засобів візуалізації мережі |
| За рівнем конкурентної боротьби: міжнародний | Ринок охоплює усі розвинені країни | Забезпечення відповідності програмного продукту широкому рівню потреб користувачів |
| За галузевою ознакою: внутрішньогалузева | Конкуренти знаходяться в одній галузі (ПЗ) | Аналіз факторів, що можуть вплинути на успіх у даній галузі |

Продовження таблиці 10.8

| | | |
|---|---|---|
| Конкуренція за видами товарів: товарно-видова | Товар одного виду - ПЗ | Проект орієнтований на середній та великий бізнес, тому необхідний вихід на ринок в даних областях |
| Особливості конкурентного середовища | В чому проявляється дана характеристика | Вплив діяльності підприємства |
| За характером конкурентних переваг: не цінова | Товар якісніший за продукти конкурентів | Зосередження ресурсів на підтримання якості продукту |
| За інтенсивністю: не марочна | Товар тільки потрапляє на ринок | Забезпечення успішного старту продукту |

Після проведення аналізу конкуренції на ринку, стає очевидним, що конкурентні товари утримують олігополію за рахунок включення можливостей системи відображення мережевої інфраструктури в якості частини більшої системи моніторингу. Даний недолік може бути використаний проектом для укріплення позицій на ринку. Аналіз конкуренції в галузі за М. Портером представлено на таблиці 10.9.

Аналіз конкуренції за М.Портером показав, що на даний момент в часі існують лише безпосередні конкуренти, шанси виникнення потенційних клієнтів є малими, товарів-замінників не існує. Отже, конкуренція на ринку є сприятливою і може мати успіх. Проте, для цього необхідно визначити ключові сильні сторони проекту, які забезпечать конкурентоспроможість. Обґрунтування факторів конкурентоспроможності зображено в таблиці 10.10.

Таблиця 10.9 – Аналіз конкуренції в галузі за М.Портером

| | Прямі конкуренти в галузі | Потенційні конкуренти | Постачальники | Клієнти | Товари-замінники |
|---|---|---|---|---|---|
| Складові аналізу | Системи моніторингу Zabbix, LanState, The Dude | Конкурентом може стати продукт в сфері візуалізації стану мережі | Постачальники послуг, пов’язаних з використанням хмарних сервісів, | Компанії, що викор-ть системи м-гу | Окрім конкурентів товарів-замінників не існує |
| В-ки | Конкуренція існує, проте досить низька | Шанс появи потенційних конкурентів доволі малий | Існує невелика залежність від пост-ів, проте вони не диктують умови роботи продукту | Головна умова - інтегра-ція з система-ми моніт-гу | Обмежень на ринку через товари-замінники не існує. |

Таблиця 10.10 – Обґрунтування факторів конкурентоспроможності

| № | Фактор | Обґрунтування факторів |
|---|---|---|
| 1. | Зручність використання | Продукт надає дружелюбний інтерфейс для спрощення процесу спостереження за станом мережі |
| 2. | Інтеграція з системами моніторингу | Наявність засобів використання з іншими системами дозволяє не прив’язуватись до конкретної системи моніторингу |

Продовження таблиці 10.10

| | | |
|---|---|---|
| 3. | Автоматична побудова графу мережевої інфраструктури | Автоматизація процесу побудови графу мережі дозволяє зменшити час на налаштування процесу спостереження за станом мережі |
| 4. | Оновлення графу в реальному часі | Дозволяє спостерігати зміни мережевої інфраструктури в реальному часі |

Для того, щоб мати успіх в сучасному ринковому середовищі, необхідно визначити ряд обраних факторів, на основі яких буде засновано перевагу продукту над конкурентами. Найважливішими серед них є інтеграція з системами моніторингу та оновлення графу в реальному часі. Оскільки основні конкуренти представляють засоби візуалізації стану мережі у вигляді частини моніторингу, новий продукт повинен мати можливість взаємодії з існуючими системами. На основі визначених факторів конкурентоспроможності було проведено аналіз сильних та слабких сторін стартап-проекту (табл. 10.11).

Табл. 10.11 – Порівняльний аналіз сильних і слабких сторін стартап-проекту

| Фактор конкурентоспроможності | Бали 1-20 | Рейтинг товарів-конкурентів порівняно з проектом | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | -3 | -2 | -1 | 0 | +1 | +2 | +3 |
| Зручність використання | 10 | TheDude | Zabbix | | LanState | | | |
| Інтеграція з системами моніторингу | 12 | TheDude, Zabbix, LanState | | | | | | |
| Автоматична побудова графу мережі | 14 | Zabbix | The Dude | | LanState | | | |
| Оновлення графу в реальному часі | 8 | Zabbix | The Dude | | LanState | | | |

Порівняльний аналіз продемонстрував, що за факторами конкурентоспроможності, стартап-проект має вагомі переваги над конкурентами і тому має шанси на успіх на ринку. Наступним кроком є складання SWOT-аналізу на основі ринкових можливостей та загроз, а також сильних і слабких сторін (табл.10.12).

Таблиця 10.12 – SWOT-аналіз стартап-проекту

| | |
|---|---|
| Сильні сторони:<br>- новизна проекту;<br>- дружелюбний інтерфейс;<br>- широкі засоби інтеграції з існуючими системами моніторингу;<br>- автоматичне формування графу мережевої інфраструктури;<br>- оновлення графу в реальному часі. | Слабкі сторони:<br>- необхідність побудови та підтримки стрімерів для взаємодії з іншими системами;<br>- складність підтримки за відсутності рішення для моніторингу;<br>- відносна залежність від постачальників. |
| Можливості:<br>- розширення підтримуваних систем через співпрацю з виробниками систем моніторингу;<br>- поява нових технологій здійснення моніторингу мережі;<br>- послаблення позицій конкурентів;<br>- поява інвесторів. | Загрози:<br>- використання клієнтами непідтримуваних систем моніторингу;<br>- відсутність розгорнутої системи моніторингу в середовищі клієнтів;<br>- поява нових сильних конкурентів; |

Після проведення SWOT-аналізу, були розроблені можливі альтернативи ринкової поведінки для виведення стартап-проекту на ринок (табл.10.13). SWOT-аналіз дав можливість детально оцінити і порівняти ключові особливості проекту та ринкового середовища.

Таблиця 10.13 – Альтернативи ринкового впровадження

| Альтернатива (орієнтовний комплекс заходів) ринкової поведінки | Ймовірність отримання ресурсів | Строки реалізації |
|---|---|---|
| Наступник | Суттєва | Максимум 1 рік |
| Загарбник | Значна | Максимум 2 роки |

Для даного стартап-проекту були обрані дві альтернативи ринкової поведінки – наступник та загарбник. Наступник є оптимальною альтернативою, оскільки проект орієнтований на досить великий сегмент ринку, на якому конкуренція представлена у вигляді олігополії. Використання даної альтернативи дозволить інтегруватися з вже діючими технологіями, а потім поступово їх витіснити. Ще однією альтернативою було обрано загарбника, який орієнтований на захоплення ринку від конкурентів.

## 10.4 Розроблення ринкової стратегії проекту

Для досягнення основної мети стартап-проекту в ринковому середовищі, необхідно розробити ринкову стратегію з набору заходів, спрямованих на отримання запланованих обсягів продажу та прибутку. Першим кроком визначається стратегія охоплення ринку: опис цільових груп можливих споживачів (табл. 10.14).

Серед потенційних споживачів були розглянуті компанії в залежності від того, чи вони використовують системи, що здійснюють моніторинг мережі. Основний пріоритет було віддано компаніям, що вже мають розгорнуту систему моніторингу, оскільки це значно спрощує інтеграцію та розгортання продукту. Наступним кроком побудови ринкової стратегії є визначення базової стратегії розвитку (табл. 10.15).

Таблиця 10.14 – Вибір цільових груп потенційних споживачів

| № | Опис профілю цільової групи потенційних клієнтів | Готовність споживачів сприйняти продукт | Орієнтований попит в межах цільової групи (сегменту) | Інтенсивність конкуренції в сегменті | Простота входу в сегмент |
|---|---|---|---|---|---|
| 1. | Компанії-користувачі систем моніторингу | Висока | Високий | Середня | Середня |
| 2. | Компанії, що не використовують засоби моніторингу мереж | Висока | Високий | Низька | Висока |
| Які цільові групи обрано: компанії-користувачі систем моніторингу та компанії, що не використовують систем моніторингу мереж. | | | | | |

Таблиця 10.15 – Визначення базової стратегії розвитку

| Обрана альтернатива розвитку ринку | Стратегія охоплення ринку | Ключові конкурентоспроможні позиції відповідно до обраної альтернативи | Базова стратегія розвитку |
|---|---|---|---|
| Наступник | Концентрація на потребах одного цільового сегменту ринку | Надання клієнтам програмного засобу відображення стану мережевої інф-и | Стратегія спеціалізації |

З можливих базових стратегій була обрана стратегія спеціалізації у якості найбільш оптимальної. Дана стратегія передбачає концентрацію на одному сегменті ринку, і задоволенню потреб цього сегменту на вищому порівняно з конкурентами рівні. Оскільки головною ідеєю стартапу є створення самостійної системи відображення стану мережевої інфраструктури, обрана стратегія є підходящою. Наступним пунктом в розробці ринкової стратегії є вибір стратегій конкурентної поведінки (табл. 10.16).

Таблиця 10.16 – Визначення базової стратегії конкурентної поведінки

| Чи є проект «першопрохідцем» на ринку? | Чи буде компанія шукати нових споживачів, або забирати вже існуючих у конкурентів? | Чи буде компанія копіювати основні характеристики товару конкурента, і які? | Стратегія конкурентної поведінки |
|---|---|---|---|
| Проект є наступником вже існуючих рішень | В першу чергу необхідно витіснити вже існуючих конкурентів, а вже потім шукати нових споживачів | Копіювання наступних характеристик:<br>- автоматична побудова карти мережі;<br>- оновлення відображення в реальному часі. | Стратегія виклику лідера |

З існуючих стратегій конкурентної поведінки була обрана стратегія виклику лідера. Вона полягає у здійсненні активної конкуренції з лідерами ринку для того, щоб зайняти їх місце. Даний підхід виправданий конкурентним середовищем ринку (олігополія), а також перевагами продукту порівняно з конкурентами. Тому така

стратегія є обґрунтованою, хоча і може мати певний ризик. Наступним кроком є визначення стратегії позиціонування (табл. 10.18).

Таблиця 10.18 – Визначення стратегії позиціонування

| | | | |
|---|---|---|---|
| Вимогу до товару цільової аудиторії | Базова стратегія розвитку | Ключові конкурентоспроможні позиції власного стартап-проекту | Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових) |
| Засоби інтеграції, автоматична побудова карти мережі, зручний інтерфейс | Стратегія спеціалізації | Стратегія виклику лідера | Можливість розширення системи, легкість інтеграції, гнучка модель конфігурації |

Для визначення стратегії позиціонування було використано основні сильні сторони стартап-проекту, завдяки яким можна здобути перевагу над конкурентами, які були також враховані серед факторів конкурентоспроможності.

## 10.5 Розроблення маркетингової стратегії стартап-проекту

Для успішної реалізації стартап-проекту необхідно розробити маркетингову програму, набір завдань, виробничого, організаційного плану, з визначенням ресурсів, які будуть використані. Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього необхідно визначити ключові переваги концепції потенційного товару (табл. 10.18).

Таблиця 10.18 – визначення ключових переваг концепції потенційного товару

| Потреба | Вигода, яку пропонує товар | Ключові переваги над конкурентами (існуючі або такі, що потрібно створити) |
|---|---|---|
| Потреба в якісному програмному забезпеченні для відображення стану мережі | Сучасний програмний продукт, що має вагомі переваги перед аналогами | Кроссплатформеність додатку, легкість у використанні, автоматична побудова графу мережевої інфраструктури |
| Потреба в спостереженні за змінами в стані в мережі | Наявні засоби для інтеграції продукту з існуючими рішеннями | Легкість в інтеграції продукту з системами моніторингу, автоматичне відображення оновлень при зміні в графі |

Основними перевагами концепції є зручність у користуванні графічний інтерфейс клієнта, а також легкість інтеграції з існуючими та перевіреними рішеннями. Далі необхідно розробити трирівневу маркетингову модель товару (табл. 10.19).

Таблиця 10.19 – Опис трьох рівнів моделі товару

| Рівні товару | Сутність та складові |
|---|---|
| | Опис базової потреби споживача, яку задовольняє товар (згідно концепції), її основної функціональної вигоди |
| | Надання програмного забезпечення для відображення стану мережевої інфраструктури в реальному часі |

Продовження таблиці 10.19

| | Властивості/Характеристики | М/Нм | Вр/Тх/Тл/Е/Ор |
|---|---|---|---|
| II. Послуга в реальному часі | 1. Економічні: зменшення ціни на обслуговування та експлуатацію<br>2. Технологічні: використання сучасних технологій<br>3. Ергономічність: зручний користувацький інтерфейс, зрозумілий функціонал продукту<br>4. Естетичні: привабливий та якісний дизайн продукту<br>5. Безпеки: відповідність нормативам використання електронних пристроїв та програмного забезпечення<br>6. Екологічні: відповідність нормативам використання програмного забезпечення | -/+ | +/+/+/+/+ |
| | Якість: використання відкритих стандартів для взаємодії з повідомленнями про проблеми об’єкту мережі (alarms) | | |
| | Документи виконані з логотипом підприємства | | |
| | Марка: nvt | | |
| | До продажу: представлення клієнтам продукту | | |
| | Після продажу: допомога в налаштуванні для середовища клієнта, підтримка продукту | | |
| За рахунок чого потенційний товар буде захищено від копіювання: захищення інтелектуальної власності шляхом патентування | | | |

Після формування маркетингової моделі товару наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар.

Таблиця 10.20 – Визначення меж встановлених цін

| Рівень цін на товари-замінники | Рівень цін на товари-аналоги | Рівень доходів цільової групи споживачів | Верхня та нижня межа встановлення ціни на товар/послугу |
|---|---|---|---|
| 800-1000 ум.од. | 800-1000 ум.од. | 4800-5500 ум.од. | 400-500 ум.од. |

В розрахунках орієнтованої ціни було також включено ціну послуги постачальників. Таким чином, потенціний товар є доступним для цільової групи споживачів, проте є дешевшим за конкурентів. Наступним кроком є формування системи збуту (табл. 10.21).

Таблиця 10.21 – Формування системи збуту

| Специфіка закупівельної поведінки цільових клієнтів | Функції збуту, які має виконувати постачальник товару | Глибина каналу збуту | Оптимальна система збуту |
|---|---|---|---|
| Замолення системи та стрімера для інфраструктури | Швидкість, надійність, підтримка проекту | Глибока | Власні сили, партнерська взаємодія з розробниками систем моніторингу |

Для оптимальної систему збуту було вирішено проводити збут власними силами, розповсюджуючи товар на умові замовлень. Проте, також існує можливість співпраці з розробниками систем моніторингу. Останньою складовою маркетингової програми є розроблення маркетингових комунікацій (табл. 10.22).

Таблиця 10.22 – концепція маркетингових комунікацій

| Специфіка поведінки цільових клієнтів | Канали комунікацій, якими користуються цільові клієнти | Ключові позиції, обрані для позиціонування | Завдання рекламного повідомлення | Концепція рекламного звернення |
|---|---|---|---|---|
| Цільові клієнти використовують Інтернет, проте також їх представники відвідують тематичні конференції | Інтернет, тематичні конференції, виставки | В мережі Інтернет буде здійснюватися технічна розсилка, публікуватися статті опису продукту на технічних порталах, буде проводитись демонстрація продукту на тематичних конференціяї | Донести до користувачів інформацію про більш зручний та сучасний аналог існуючих систем | Рекламне звернення повідомляє про появу нового продукту на ринку з описом його переваг над існуючими рішеннями |

Основними каналами комунікації, якими користуються цільові клієнти, є Інтернет та тематичні конференції, тому їх використання є виправданим. Також, публікування на технічних ресурсах дозволить зацікавити фахівців, розширити обсяг цільової аудиторії та залучити більше нових клієнтів.

## 10.6 Висновки до розділу

В даному розділі було розглянуто потенціал створення розроблюваної системи в якості стартап-проекту, обгрунтовано її конкурентоспроможність, проведено аналіз можливих шляхів виходу на ринок, досліджено існуючі стратегії розвитку та маркетингу.

Були сформовані основні техніко-економічні характеристики ідеї, її особливості в порівнянні з існуючими аналогами. З них були виділені: забезпечення інтеграції з вже існуючими системами моніторингу, автоматична побудова графу мережі, відображення змін в реальному часі та зручний в роботі інтерфейс. При порівнянні даних характеристик з аналогами було встановлено, що ідея є конкурентоспроможною, і може мати потенційний успіх на ринку.

Також було проведено технологічний аудит, в якому було перевірено технології, які могли бути використаними для реалізації проекту. Перелік основних технологій включає: графова БД ArangoDB, мова програмування серверної частини Go, мова програмування клієнтської частини JavaScript, фреймворк для побудови графічних інтерфейсів VueJS. Використання даних технологій є доступним для даного проекту, оскільки всі вони викладені у відкритому доступі.

Було проведено аналіз ринкових можливостей запуску стартап-проекту. Для цього були розглянуті: характеристика потенційного ринку, стан конкуренції на ринку, ринкове впровадження стартап-проекту. Згідно з проведеним аналізом, було встановлено, що проект має вагомі переваги над аналогами, а отже є конкурентоспроможним. При цьому для проекту також були обрані альтернативи ринкової поведінки.

Також були розроблені ринкові стратегії проекту. Для цього було проаналізована потенційні групи клієнтів, серед яких основний акцент було обрано на компаніях, що вже мають у розпорядженні активну систему моніторингу мережі. Обрано спеціалізацію та стратегію конкурентної поведінки виклику лідера, з врахуванням особливостей даного сегменту ринку. Оскільки даний продукт орієнтовано на досить великий сегмент ринку, а також беручи до уваги конкурентну олігополію, використання даної стратегії є виправданим.

Було розроблено маркетингову програму стартап-проекту, в результаті чого були встановлені цінові межі на потенційний продукт та обрані джерела збуту. Пріоритетом маркетингової програми є встановлення низьких цін для обраного сегменту клієнтів, а також обрання підходящих каналів збуту. Саме тому було обрано Інтернет та тематичні конференції, оскільки дані канали комунікації є найбільш використовуваними серед ключової групи клієнтів, а отже дозволять охопити найбільшу аудиторію.

# ВИСНОВКИ

Результатом магістерської дисертації є розроблена система для відображення змін в стані мережевої інфраструктури в реальному часі. Були досліджені характерні особливості існуючих програмних продуктів в даній сфері, їх основний функціонал, набір проблем, які вони вирішують та зручність їх використання. Під час дослідження систем візуалізації мережі було виявлено, що всі розглянуті рішення є частинами систем моніторингу мереж.

На основі проведеного аналізу існуючих продуктів була сформована концепція представленої в роботі системи, утворено вимоги до розроблюваної системи, визначено основні переваги та шляхи вдосконалення існуючого функціоналу в сучасних системах візуалізації мережевої інфраструктури. Основними вимогами до системи було обрано легкість інтеграції з існуючими системами моніторингу та збору інформації про стан мережі для забезпечення плавного переходу користувачем з систем, що він використовує на даний момент; оновлення візуального представлення мережевої інфраструктури в реальному часі.

На основні сформованих вимог були встановлені ролі користувачів системи та основні сценарії використання системи. Кожна роль користувача та сценарії взаємодії з системою відповідають частині системи, з якої взаємодія користувач.

Згідно описаного набору функціоналу в сценаріях використання системи, було проведено проектування загальної архітектури системи. В результаті даного етапу розробки було виділено три основні групи компонентів: компоненти, що відповідають за отримання даних з висхідних систем та побудову графу мережевої інфраструктури; компоненти, які забезпечують відображення графа, даних про його елементи; і нарешті, компоненти підсистеми управління конфігурацією системи. Результатом проектування загальної архітектури системи стала структурна схема системи.

Після проектування загальної архітектури системи, проведено відбір технологій, які були використані для реалізації системи. Для реалізації серверної

частини системи було обрано мову програмування Go через ефективність використання ресурсів систему та широкий набір бібліотек для розробки розподілених мережевих застосунків. Проведено порівняльний аналіз рішень для обміну інформації між клієнтом та сервером в реальному часі, внаслідок чого було обрано технологію WebSocket для забезпечення передачі повідомлень про оновлення графу в реальному часі.

Залежно від пред'явлених вимог до кожної з підсистем було обрано відповідні системи для зберігання даних. Структура мережевої інфраструктури оптимально підходить під модель графових баз даних, саме тому було обрано базу ArangoDB. Для підсистеми конфігурації більш підходящою моделлю даних було визначено реляційну модель, в якості бази даних було обрано базу PostgreSQL.

Були розглянуті основні питання, які виникли при проектуванні бізнес-логіки системи, описані основні алгоритми, які лежать в основі процесу побудови графу мережевої інфраструктури. Наведено основні аспекти проектування користувацького інтерфейсу клієнтської частини системи. Представлено опис процесу тестування системи.

В розділі опису стартап-проекту було сформовано основні сильні, слабкі та нейтральні сторони програмного продукту. Проведено технологічний аудит проекту, в результаті якого була встановлена доступність всіх необхідних технологій для використання. Визначено цільову аудиторію системи та основні комунікаційні канали у вигляді мережі Інтернет та тематичних конференцій.

Розроблена система має можливість в подальшому вдосконалюватися. Одним з можливих шляхів покращення є забезпечення програмного управління сервісами-стрімерами через підсистему конфігурації. Це дозволить спростити процедуру розгортання інфраструктури для отримання даних з зовнішніх систем.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Полторак В.П. Теорія інформації та кодування / Ю.П. Жураковський, В.П. Полторак // Підручник. – К.: Вища школа, 2001. – 255 с.: іл., укр.мовою (Гриф МОНУ №1/11-2367 від 21.05.2001)."
2. Полторак В.П. Криптографічний захист даних в цифрових інформаційних системах (Частина 1) / Полторак В.П. // Телеком. Військовий звязок. Спеціальний випуск, №2/2018. - К.: Softpress. hi-Tech.ua, Жовт., 2018. - с. 98-104., Мова публікації:українська "
3. Zabbix Overview [Електронний ресурс] – https://www.zabbix.com/documentation/2.4/ru/manual/introduction/overview
4. Lan State [Мережевий ресурс] – https://www.10-strike.ru/lanstate/
5. Network Maps Overview [Електронний ресурс] – https://habr.com/ru/post/444410/
6. Efficient Programming Languages [Електронний ресурс] – https://sdevprog.blogspot.com/2018/04/overview-of-efficient-programming.html
7. Communicating Sequential Processes [Електронний ресурс] – https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf
8. AQL Guide [Електронний ресурс] – https://www.arangodb.com/docs/stable/aql
9. VueJS Architecture [Електронний ресурс] – https://vuejs.org/v2/guide
10. SSE Connection Limit [Електронний ресурс] – https://bugs.chromium.org/p/chromium/issues/detail?id=275955
11. JSON [Електронний ресурс] – https://www.json.org/json-en.html
12. Streaming Connectors [Електронний ресурс] – https://www.slideshare.net/akilaanusari/stream-connectors
13. Functional Options For Friendly APIs [Електронний ресурс] – https://dave.cheney.net/2014/10/17/functional-options-for-friendly-apis
14. PostgreSQL LISTEN/NOTIFY [Електронний ресурс] – https://tapoueh.org/blog/2018/07/postgresql-listen-notify/

15. Гамма Э. Приемы обьектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2015. – с.173;
16. Попов Д. Оптимизирующие парсер-комбинаторы / Попов Д. // Практика функционального программирования – 2010 – №5 – ст.116;
17. Marenko L. Achieving Evolvable Web-Database Bioscience Applications Using the EAV/CR Framework: Recent Advances / Marenko L., Tosches N., Crasto C., Shepherd G., Miller P.L., Nadkarni P.M. – 2003 - №3 – с.1;
18. Material Design Principles [Електронний ресурс] – https://material.io/design/introduction/#principles
19. Степанченко И. В. Методы тестирования программного обеспечения; навч. посібник - ВолгГТУ, Волгоград, 2006. – 74 с.